
Amazon Redshift

Database Developer Guide



Amazon Redshift: Database Developer Guide

Copyright © 2023 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Introduction	1
Prerequisites	1
Are you a database developer?	1
System and architecture overview	2
Data warehouse system architecture	3
Performance	5
Columnar storage	7
Workload management	8
Using Amazon Redshift with other services	9
Best practices	10
Conducting a proof of concept	10
Overview of the process	10
Identify the business goals and success criteria	11
Set up your proof of concept	11
Checklist for a complete evaluation	12
Develop a project plan for your evaluation	13
Additional resources to help your evaluation	14
Need help?	15
Best practices for designing tables	15
Choose the best sort key	15
Choose the best distribution style	16
Use automatic compression	17
Define constraints	17
Use date/time data types for date columns	17
Best practices for loading data	17
Take the loading data tutorial	18
Use a COPY command to load data	18
Use a single COPY command	18
Loading data files	18
Compressing your data files	19
Verify data files before and after a load	19
Use a multi-row insert	19
Use a bulk insert	20
Load data in sort key order	20
Load data in sequential blocks	20
Use time-series tables	21
Use a staging table to perform a merge	21
Schedule around maintenance windows	21
Best practices for designing queries	21
Working with Advisor	23
Amazon Redshift Regions	23
Access Advisor	24
Advisor recommendations	24
Tutorials	35
Working with automatic table optimization	36
Enabling automatic table optimization	36
Removing automatic table optimization	37
Monitoring actions of automatic table optimization	37
Working with column compression	38
Compression encodings	39
Testing compression encodings	45
Example: Choosing compression encodings for the CUSTOMER table	47
Working with data distribution styles	49
Data distribution concepts	50

Distribution styles	51
Viewing distribution styles	52
Evaluating query patterns	53
Designating distribution styles	53
Evaluating the query plan	54
Query plan example	55
Distribution examples	58
Working with sort keys	60
Compound sort key	61
Interleaved sort key	61
Defining table constraints	62
Loading data	63
Using COPY to load data	63
Credentials and access permissions	64
Preparing your input data	65
Loading data from Amazon S3	66
Loading data from Amazon EMR	74
Loading data from remote hosts	78
Loading from Amazon DynamoDB	84
Verifying that the data loaded correctly	86
Validating input data	86
Automatic compression	87
Optimizing for narrow tables	89
Default values	89
Troubleshooting	90
Continuous file ingestion (preview)	94
Updating with DML	96
Updating and inserting	96
Merge method 1: Replacing existing rows	97
Merge method 2: Specifying a column list	97
Creating a temporary staging table	97
Performing a merge operation by replacing existing rows	98
Performing a merge operation by specifying a column list	99
Merge examples	100
Performing a deep copy	102
Analyzing tables	105
Automatic analyze	105
Analysis of new table data	106
ANALYZE command history	109
Vacuuming tables	110
Automatic table sort	110
Automatic vacuum delete	111
VACUUM frequency	111
Sort stage and merge stage	111
Vacuum threshold	112
Vacuum types	112
Managing vacuum times	112
Managing concurrent write operations	118
Serializable isolation	119
Write and read/write operations	122
Concurrent write examples	123
Tutorial: Loading data from Amazon S3	124
Prerequisites	124
Overview	125
Steps	125
Step 1: Create a cluster	125
Step 2: Download the data files	126

Step 3: Upload the files to an Amazon S3 bucket	127
Step 4: Create the sample tables	128
Step 5: Run the COPY commands	130
Step 6: Vacuum and analyze the database	142
Step 7: Clean up your resources	142
Summary	143
Unloading data	144
Unloading data to Amazon S3	144
Unloading encrypted data files	146
Unloading data in delimited or fixed-width format	148
Reloading unloaded data	149
Creating user-defined functions	150
UDF security and privileges	150
Creating a scalar SQL UDF	151
Scalar SQL function example	151
Creating a scalar Python UDF	152
Scalar Python UDF example	152
Python UDF data types	152
ANYELEMENT data type	153
Python language support	153
UDF constraints	156
Creating a scalar Lambda UDF	157
Registering a Lambda UDF	157
Managing Lambda UDF security and privileges	157
Configuring the authorization parameter for Lambda UDFs	158
Using the JSON interface between Amazon Redshift and Lambda	157
Naming UDFs	161
Logging errors and warnings	162
Example uses of UDFs	163
Creating stored procedures	164
Stored procedure overview	164
Naming stored procedures	166
Security and privileges	167
Returning a result set	168
Managing transactions	169
Trapping errors	178
Logging stored procedures	181
Considerations	181
PL/pgSQL language reference	182
PL/pgSQL reference conventions	182
Structure of PL/pgSQL	182
Supported PL/pgSQL statements	186
Creating materialized views	198
Querying a materialized view	200
Automatic query rewriting to use materialized views	201
Usage notes	201
Limitations	201
Refreshing a materialized view	202
Autorefreshing a materialized view	204
Automated materialized views	204
SQL scope and considerations for automated materialized views	205
Automated materialized views limitations	206
Billing for automated materialized views	206
Additional resources	206
Using a user-defined function (UDF) in a materialized view	206
Referencing a UDF in a materialized view	206
Streaming ingestion	208

Data flow	208
Streaming ingestion use cases	208
Streaming ingestion considerations	209
Limitations	209
Getting started with streaming ingestion from Amazon Kinesis Data Streams	210
Getting started with streaming ingestion from Amazon Managed Streaming for Apache Kafka ..	213
Electric vehicle station-data streaming ingestion tutorial, using Kinesis	216
Querying spatial data	220
Tutorial: Using spatial SQL functions	222
Prerequisites	222
Step 1: Create tables and load test data	223
Step 2: Query spatial data	225
Step 3: Clean up your resources	227
Loading a shapefile	228
Terminology	229
Bounding box	229
Geometric validity	229
Geometric simplicity	231
Limitations	232
Querying data with federated queries	234
Getting started with using federated queries to PostgreSQL	234
Getting started using federated queries to PostgreSQL with CloudFormation	235
Launching a CloudFormation stack for Redshift federated queries	236
Querying data from the external schema	237
Getting started with using federated queries to MySQL	237
Creating a secret and an IAM role	238
Prerequisites	238
Examples of using a federated query	240
Example of using a federated query with PostgreSQL	240
Example of using a mixed-case name	242
Example of using a federated query with MySQL	243
Data type differences	243
Considerations	246
Querying external data using Amazon Redshift Spectrum	248
Amazon Redshift Spectrum overview	248
Amazon Redshift Spectrum Regions	249
Amazon Redshift Spectrum considerations	249
Getting started with Amazon Redshift Spectrum	250
Prerequisites	250
CloudFormation	250
Getting started with Redshift Spectrum step by step	250
Step 1. Create an IAM role	251
Step 2: Associate the IAM role with your cluster	253
Step 3: Create an external schema and an external table	254
Step 4: Query your data in Amazon S3	255
Launch your CloudFormation stack and then query your data	257
IAM policies for Amazon Redshift Spectrum	259
Amazon S3 permissions	260
Cross-account Amazon S3 permissions	260
Grant or restrict access using Redshift Spectrum	261
Minimum permissions	262
Chaining IAM roles	263
Accessing AWS Glue data	263
Using Redshift Spectrum with Lake Formation	269
Using data filters for row-level and cell-level security	270
Creating data files for queries in Amazon Redshift Spectrum	270
Data formats for Redshift Spectrum	270

Compression types for Redshift Spectrum	271
Encryption for Redshift Spectrum	272
Creating external schemas	272
Working with external catalogs	274
Creating external tables	277
Pseudocolumns	278
Partitioning Redshift Spectrum external tables	279
Mapping to ORC columns	282
Creating external tables for Hudi-managed data	284
Creating external tables for Delta Lake data	285
Improving Amazon Redshift Spectrum query performance	287
Setting data handling options	289
Performing correlated subqueries	290
Monitoring metrics	290
Troubleshooting queries	291
Retries exceeded	291
Access throttled	291
Resource limit exceeded	292
No rows returned for a partitioned table	293
Not authorized error	293
Incompatible data formats	293
Syntax error when using Hive DDL in Amazon Redshift	293
Permission to create temporary tables	294
Invalid range	294
Invalid Parquet version number	294
Tutorial: Querying nested data with Amazon Redshift Spectrum	294
Overview	294
Step 1: Create an external table that contains nested data	295
Step 2: Query your nested data in Amazon S3 with SQL extensions	296
Nested data use cases	300
Nested data limitations	301
Serializing complex nested JSON	302
Using HyperLogLog sketches in Amazon Redshift	305
Considerations	305
Limitations	306
Examples	306
Example: Return cardinality in a subquery	306
Example: Return an HLLSKETCH type from combined sketches in a subquery	307
Example: Return a HyperLogLog sketch from combining multiple sketches	307
Example: Generate HyperLogLog sketches over S3 data using external tables	308
Querying data across databases	310
Considerations	311
Limitations	311
Examples of using a cross-database query	312
Using cross-database queries with the query editor	315
Sharing data across clusters	317
Regions where data sharing is available	317
Data sharing overview	318
Data sharing use cases	318
Data sharing concepts	318
Sharing data at different levels	320
Managing data consistency	320
Accessing shared data	320
Considerations when using data sharing in Amazon Redshift	320
How data sharing works	321
Controlling access for cross-account datashares	322
Working with views in Amazon Redshift data sharing	325

Managing the data sharing lifecycle	326
Managing permissions for datashares	326
Tracking usage and auditing in data sharing	327
Cluster management and data sharing	328
Integrating Amazon Redshift data sharing with business intelligence tools	328
Accessing metadata for datashares	329
Working with AWS Data Exchange for Amazon Redshift	330
How AWS Data Exchange datashares work	330
Considerations when using AWS Data Exchange for Amazon Redshift	331
AWS Lake Formation-managed Redshift datashares	331
Considerations and limitations when using AWS Lake Formation with Amazon Redshift	332
Getting started data sharing	333
Getting started data sharing using the SQL interface	333
Getting started data sharing using the console	356
Getting started data sharing with CloudFormation	365
Ingesting and querying semistructured data in Amazon Redshift	368
Use cases for the SUPER data type	368
Concepts for SUPER data type use	369
Considerations for SUPER data	370
SUPER sample dataset	370
Loading semistructured data into Amazon Redshift	372
Parsing JSON documents to SUPER columns	372
Using COPY to load JSON data in Amazon Redshift	373
Unloading semistructured data	376
Unloading semistructured data in CSV or text formats	376
Unloading semistructured data in the Parquet format	377
Querying semistructured data	377
Navigation	377
Unnesting queries	378
Object unpivoting	379
Dynamic typing	380
Lax semantics	382
Types of introspection	382
Order by	383
Operators and functions	384
Arithmetic operators	384
Arithmetic functions	384
Array functions	385
SUPER configurations	386
Lax and strict modes for SUPER	386
Accessing JSON fields with uppercase and mixedcase letters	386
Parsing options	387
Limitations	388
Using SUPER data type with materialized views	391
Accelerating PartiQL queries	391
Limitations for using the SUPER data type with materialized views	394
Using machine learning in Amazon Redshift	395
Machine learning overview	396
How machine learning can solve a problem	396
Terms and concepts for Amazon Redshift ML	397
Machine learning for novices and experts	398
Costs for using Amazon Redshift ML	399
Getting started with Amazon Redshift ML	400
Administrative setup	401
Using model explainability with Amazon Redshift ML	404
Amazon Redshift ML probability metrics	405
Tutorials for Amazon Redshift ML	406

Tuning query performance	463
Query processing	463
Query planning and execution workflow	463
Query plan	465
Reviewing query plan steps	470
Factors affecting query performance	472
Analyzing and improving queries	473
Query analysis workflow	473
Reviewing query alerts	474
Analyzing the query plan	475
Analyzing the query summary	476
Improving query performance	481
Diagnostic queries for query tuning	483
Troubleshooting queries	486
Connection fails	487
Query hangs	487
Query takes too long	488
Load fails	489
Load takes too long	489
Load data is incorrect	489
Setting the JDBC fetch size parameter	489
Implementing workload management	491
Modifying the WLM configuration	492
Migrating from manual WLM to automatic WLM	492
Automatic WLM	493
Priority	494
Concurrency scaling mode	494
User groups	494
Query groups	494
Wildcards	495
Query monitoring rules	495
Checking for automatic WLM	495
Query priority	495
Manual WLM	499
Concurrency scaling mode	500
Concurrency level	500
User groups	501
Query groups	501
Wildcards	501
WLM memory percent to use	501
WLM timeout	502
Query monitoring rules	502
WLM query queue hopping	502
Tutorial: Configuring manual WLM queues	505
Concurrency scaling	515
Concurrency scaling capabilities	515
Limitations for concurrency scaling	516
Regions for concurrency scaling	516
Concurrency scaling candidates	517
Configuring concurrency scaling queues	496
Monitoring concurrency scaling	517
Concurrency scaling system views	518
Short query acceleration	518
Maximum SQA runtime	519
Monitoring SQA	519
WLM queue assignment rules	520
Queue assignments example	521

Assigning queries to queues	523
Assigning queries to queues based on user groups	523
Assigning a query to a query group	523
Assigning queries to the superuser queue	523
Dynamic and static properties	524
WLM dynamic memory allocation	525
Dynamic WLM example	525
Query monitoring rules	527
Defining a query monitor rule	527
Query monitoring metrics for Amazon Redshift	528
Query monitoring metrics for Amazon Redshift Serverless	530
Query monitoring rules templates	532
System tables and views for query monitoring rules	532
WLM system tables and views	533
WLM service class IDs	534
Managing database security	535
Amazon Redshift security overview	535
Default database user permissions	536
Superusers	537
Users	537
Creating, altering, and deleting users	537
Groups	538
Creating, altering, and deleting groups	538
Example for controlling user and group access	538
Schemas	539
Creating, altering, and deleting schemas	540
Search path	540
Schema-based permissions	541
Role-based access control	541
Role hierarchy	541
Role assignment	542
Amazon Redshift system-defined roles	542
System permissions	543
Database object permissions	546
ALTER DEFAULT PRIVILEGES for RBAC	546
Considerations for role usage	546
Managing roles	547
Row-level security	547
Using RLS policies in SQL statements	547
Combining multiple policies per user	547
RLS policy ownership and management	548
Policy-dependent objects and principles	548
Considerations	550
Best practices for RLS performance	551
Creating, attaching, detaching, and dropping RLS policies	552
Dynamic data masking	555
Overview	555
End-to-end example	556
Considerations when using dynamic data masking	558
Managing dynamic data masking policies	559
Masking policy hierarchy	560
Conditional dynamic data masking	561
System views for dynamic data masking	562
SQL reference	564
Amazon Redshift SQL	564
SQL functions supported on the leader node	564
Amazon Redshift and PostgreSQL	565

Using SQL	570
SQL reference conventions	571
Basic elements	571
Expressions	601
Conditions	605
SQL commands	623
ABORT	626
ALTER DATABASE	627
ALTER DATASHARE	629
ALTER DEFAULT PRIVILEGES	631
ALTER GROUP	633
ALTER IDENTITY PROVIDER	634
ALTER MASKING POLICY	635
ALTER MATERIALIZED VIEW	636
ALTER ROLE	636
ALTER PROCEDURE	637
ALTER SCHEMA	638
ALTER TABLE	640
ALTER TABLE APPEND	656
ALTER USER	659
ANALYZE	663
ANALYZE COMPRESSION	665
ATTACH MASKING POLICY	667
ATTACH RLS POLICY	668
BEGIN	669
CALL	670
CANCEL	672
CLOSE	674
COMMENT	674
COMMIT	676
COPY	677
CREATE DATABASE	745
CREATE DATASHARE	754
CREATE EXTERNAL FUNCTION	755
CREATE EXTERNAL SCHEMA	762
CREATE EXTERNAL TABLE	768
CREATE FUNCTION	788
CREATE GROUP	792
CREATE IDENTITY PROVIDER	792
CREATE LIBRARY	793
CREATE MASKING POLICY	796
CREATE MATERIALIZED VIEW	796
CREATE MODEL	800
CREATE PROCEDURE	815
CREATE RLS POLICY	819
CREATE ROLE	820
CREATE SCHEMA	821
CREATE TABLE	823
CREATE TABLE AS	838
CREATE USER	846
CREATE VIEW	851
DEALLOCATE	854
DECLARE	855
DELETE	858
DESC DATASHARE	860
DESC IDENTITY PROVIDER	861
DETACH MASKING POLICY	861

DETACH RLS POLICY	862
DROP DATABASE	863
DROP DATASHARE	864
DROP FUNCTION	865
DROP GROUP	866
DROP IDENTITY PROVIDER	867
DROP LIBRARY	867
DROP MASKING POLICY	868
DROP MODEL	868
DROP MATERIALIZED VIEW	869
DROP PROCEDURE	870
DROP RLS POLICY	870
DROP ROLE	871
DROP SCHEMA	872
DROP TABLE	874
DROP USER	876
DROP VIEW	878
END	879
EXECUTE	880
EXPLAIN	881
FETCH	886
GRANT	887
INSERT	902
INSERT (external table)	906
LOCK	908
MERGE	909
PREPARE	912
REFRESH MATERIALIZED VIEW	914
RESET	915
REVOKE	916
ROLLBACK	927
SELECT	928
SELECT INTO	976
SET	977
SET SESSION AUTHORIZATION	980
SET SESSION CHARACTERISTICS	981
SHOW	981
SHOW EXTERNAL TABLE	982
SHOW DATABASES	984
SHOW MODEL	984
SHOW DATASHARES	986
SHOW PROCEDURE	987
SHOW TABLE	988
SHOW VIEW	989
START TRANSACTION	990
TRUNCATE	990
UNLOAD	991
UPDATE	1010
VACUUM	1015
SQL functions reference	1020
Leader node-only functions	1020
Compute node-only functions	1021
Aggregate functions	1022
Array functions	1041
VARBYTE functions	1044
Bit-wise aggregate functions	1051
Conditional expressions	1056

Data type formatting functions	1066
Date and time functions	1086
Hash functions	1131
HyperLogLog functions	1138
JSON functions	1142
Machine learning functions	1152
Math functions	1154
Object functions	1179
Spatial functions	1180
String functions	1277
SUPER type information functions	1325
Window functions	1335
System administration functions	1379
System information functions	1386
Reserved words	1405
System tables and views reference	1409
System tables and views	1409
Types of system tables and views	1409
Visibility of data in system tables and views	1410
Filtering system-generated queries	1410
SVV metadata views	1411
SVV_ALL_COLUMNS	1412
SVV_ALL_SCHEMAS	1413
SVV_ALL_TABLES	1414
SVV_ALTER_TABLE_RECOMMENDATIONS	1415
SVV_ATTACHED_MASKING_POLICY	1416
SVV_COLUMNS	1418
SVV_COLUMN_PRIVILEGES	1419
SVV_DATABASE_PRIVILEGES	1420
SVV_DATASHARE_PRIVILEGES	1421
SVV_DATASHARES	1422
SVV_DATASHARE_CONSUMERS	1423
SVV_DATASHARE_OBJECTS	1424
SVV_DEFAULT_PRIVILEGES	1425
SVV_DISKUSAGE	1426
SVV_EXTERNAL_COLUMNS	1428
SVV_EXTERNAL_DATABASES	1429
SVV_EXTERNAL_PARTITIONS	1429
SVV_EXTERNAL_SCHEMAS	1430
SVV_EXTERNAL_TABLES	1431
SVV_FUNCTION_PRIVILEGES	1432
SVV_IAM_PRIVILEGES	1433
SVV_INTERLEAVED_COLUMNS	1434
SVV_IDENTITY_PROVIDERS	1434
SVV_LANGUAGE_PRIVILEGES	1435
SVV_MASKING_POLICY	1436
SVV_ML_MODEL_INFO	1437
SVV_ML_MODEL_PRIVILEGES	1438
SVV_MV_DEPENDENCY	1439
SVV_MV_INFO	1439
SVV_QUERY_INFLIGHT	1441
SVV_QUERY_STATE	1442
SVV_REDSHIFT_COLUMNS	1444
SVV_REDSHIFT_DATABASES	1445
SVV_REDSHIFT_FUNCTIONS	1446
SVV_REDSHIFT_SCHEMAS	1447
SVV_REDSHIFT_TABLES	1448

SVV_RELATION_PRIVILEGES	1449
SVV_RLS_APPLIED_POLICY	1449
SVV_RLS_ATTACHED_POLICY	1450
SVV_RLS_POLICY	1451
SVV_RLS_RELATION	1452
SVV_ROLES	1453
SVV_ROLE_GRANTS	1454
SVV_SCHEMA_QUOTA_STATE	1454
SVV_SCHEMA_PRIVILEGES	1455
SVV_SYSTEM_PRIVILEGES	1456
SVV_TABLES	1456
SVV_TABLE_INFO	1457
SVV_TRANSACTIONS	1460
SVV_USER_INFO	1461
SVV_USER_GRANTS	1462
SVV_VACUUM_PROGRESS	1463
SVV_VACUUM_SUMMARY	1464
SYS monitoring views	1465
SYS_CONNECTION_LOG	1466
SYS_COPY_JOB (preview)	1468
SYS_DATASHARE_CHANGE_LOG	1469
SYS_DATASHARE_USAGE_CONSUMER	1471
SYS_DATASHARE_USAGE_PRODUCER	1471
SYS_EXTERNAL_QUERY_DETAIL	1472
SYS_LOAD_DETAIL	1474
SYS_LOAD_ERROR_DETAIL	1476
SYS_LOAD_HISTORY	1477
SYS_QUERY_DETAIL	1480
SYS_QUERY_HISTORY	1483
SYS_SERVERLESS_USAGE	1488
SYS_STREAM_SCAN_ERRORS	1490
SYS_STREAM_SCAN_STATES	1490
SYS_UNLOAD_DETAIL	1492
SYS_UNLOAD_HISTORY	1493
System monitoring (provisioned only)	1494
STL views for logging	1494
STV tables for snapshot data	1577
SVCS views for main and concurrency scaling clusters	1610
SVL views for main cluster	1628
System catalog tables	1670
PG_ATTRIBUTE_INFO	1671
PG_CLASS_INFO	1671
PG_DATABASE_INFO	1672
PG_DEFAULT_ACL	1673
PG_EXTERNAL_SCHEMA	1674
PG_LIBRARY	1675
PG_PROC_INFO	1676
PG_STATISTIC_INDICATOR	1676
PG_TABLE_DEF	1677
PG_USER_INFO	1679
Querying the catalog tables	1680
Configuration reference	1685
Modifying the server configuration	1685
.....	1685
analyze_threshold_percent	1685
Values (default in bold)	1685
Description	1685

Examples	1686
cast_super_null_on_error	1686
Values (default in bold)	1686
Description	1686
datashare_break_glass_session_var	1686
Values (default in bold)	1686
Description	1686
Example	1687
datestyle	1687
Values (default in bold)	1687
Description	1686
Example	1687
default_geometry_encoding	1687
Values (default in bold)	1687
Description	1686
describe_field_name_in_uppercase	1687
Values (default in bold)	1687
Description	1686
Example	1687
downcase_delimited_identifier	1688
Values (default in bold)	1688
Description	1686
enable_case_sensitive_identifier	1688
Values (default in bold)	1688
Description	1688
Examples	1689
Usage Notes	1690
enable_case_sensitive_super_attribute	1690
Values (default in bold)	1690
Description	1690
Examples	1690
Usage Notes	1691
enable_result_cache_for_session	1691
Values (default in bold)	1691
Description	1691
Example	1691
enable_vacuum_boost	1692
Values (default in bold)	1692
Description	1686
error_on_nondeterministic_update	1692
Values (default in bold)	1692
Description	1686
Example	1687
extra_float_digits	1692
Values (default in bold)	1692
Description	1692
json_serialization_enable	1693
Values (default in bold)	1693
Description	1686
json_serialization_parse_nested_strings	1693
Values (default in bold)	1693
Description	1686
max_concurrency_scaling_clusters	1693
Values (default in bold)	1693
Description	1693
max_cursor_result_set_size	1694
Values (default in bold)	1694

Description	1694
mv_enable_aqmv_for_session	1694
Values (default in bold)	1694
Description	1694
navigate_super_null_on_error	1694
Values (default in bold)	1694
Description	1686
parse_super_null_on_error	1694
Values (default in bold)	1694
Description	1686
pg_federation_repeatable_read	1695
Values (default in bold)	1695
Description	1686
Examples	1695
query_group	1695
Values (default in bold)	1695
Description	1695
search_path	1696
Values (default in bold)	1696
Description	1696
Example	1696
spectrum_enable_pseudo_columns	1697
Values (default in bold)	1697
Description	1697
Example	1697
enable_spectrum_oid	1698
Values (default in bold)	1698
Description	1698
Example	1698
spectrum_query_maxerror	1698
Values (default in bold)	1698
Description	1698
Example	1698
statement_timeout	1699
Values (default in bold)	1699
Description	1699
Example	1699
stored_proc_log_min_messages	1699
Values (default in bold)	1699
Description	1686
timezone	1700
Values (default in bold)	1700
Syntax	1700
Description	1700
Time zone formats	1700
Examples	1702
wlm_query_slot_count	1702
Values (default in bold)	1702
Description	1702
Examples	1703
Sample database	1704
CATEGORY table	1705
DATE table	1705
EVENT table	1706
VENUE table	1706
USERS table	1707
LISTING table	1707

SALES table	1708
Time zone names and abbreviations	1709
Time zone names	1709
Time zone abbreviations	1709
Document history	1711
Earlier updates	1717

Introduction

Welcome to the *Amazon Redshift Database Developer Guide*. Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. Amazon Redshift Serverless lets you access and analyze data without the usual configurations of a provisioned data warehouse. Resources are automatically provisioned and data warehouse capacity is intelligently scaled to deliver fast performance for even the most demanding and unpredictable workloads. You don't incur charges when the data warehouse is idle, so you only pay for what you use. Regardless of the size of the dataset, you can load data and start querying right away in the Amazon Redshift query editor v2 or in your favorite business intelligence (BI) tool. Enjoy the best price performance and familiar SQL features in an easy-to-use, zero administration environment.

This guide focuses on using Amazon Redshift to create and manage a data warehouse. If you work with databases as a designer, software developer, or administrator, it gives you the information you need to design, build, query, and maintain your data warehouse.

Topics

- [Prerequisites \(p. 1\)](#)
- [Are you a database developer? \(p. 1\)](#)
- [System and architecture overview \(p. 2\)](#)

Prerequisites

Before you use this guide, you should read [Amazon Redshift Serverless](#), which goes over how to complete the following tasks.

- Create a data warehouse with Amazon Redshift Serverless.
- Loading in sample data with Amazon Redshift query editor v2
- Loading in data from Amazon S3.

You should also know how to use your SQL client and should have a fundamental understanding of the SQL language.

Are you a database developer?

If you are a first-time Amazon Redshift user, we recommend you read [Amazon Redshift Serverless](#) to learn how to get started.

If you are a database user, database designer, database developer, or database administrator, the following table will help you find what you're looking for.

If you want to...	We recommend...
Learn about the internal architecture of the Amazon Redshift data warehouse.	The System and architecture overview (p. 2) gives a high-level overview of Amazon Redshift's internal architecture. If you want a broader overview of the Amazon Redshift web service, go to the Amazon Redshift product detail page.

If you want to...	We recommend...
Create databases, tables, users, and other database objects.	<p>Getting started using databases is a quick introduction to the basics of SQL development.</p> <p>The Amazon Redshift SQL (p. 564) has the syntax and examples for Amazon Redshift SQL commands and functions and other SQL elements.</p> <p>Amazon Redshift best practices for designing tables (p. 15) provides a summary of our recommendations for choosing sort keys, distribution keys, and compression encodings.</p>
Learn how to design tables for optimum performance.	<p>Working with automatic table optimization (p. 36) details considerations for applying compression to the data in table columns and choosing distribution and sort keys.</p>
Load data.	<p>Loading data (p. 63) explains the procedures for loading large datasets from Amazon DynamoDB tables or from flat files stored in Amazon S3 buckets.</p> <p>Amazon Redshift best practices for loading data (p. 17) provides for tips for loading your data quickly and effectively.</p>
Manage users, groups, and database security.	Managing database security (p. 535) covers database security topics.
Monitor and optimize system performance.	<p>The System tables and views reference (p. 1409) details system tables and views that you can query for the status of the database and monitor queries and processes.</p> <p>Also consult the Amazon Redshift Management Guide to learn how to use the AWS Management Console to check the system health, monitor metrics, and back up and restore clusters.</p>
Analyze and report information from very large datasets.	<p>Many popular software vendors are certifying Amazon Redshift with their offerings to enable you to continue to use the tools you use today. For more information, see the Amazon Redshift partner page.</p> <p>The SQL reference (p. 564) has all the details for the SQL expressions, commands, and functions Amazon Redshift supports.</p>
Interact with Amazon Redshift resources and tables.	See the Amazon Redshift Serverless API guide , the Amazon Redshift API guide , and the Amazon Redshift Data API guide to learn more about how you can programmatically interact with resources and run operations.
Follow a tutorial to become more familiar with Amazon Redshift.	Follow a tutorial in Tutorials for Amazon Redshift to learn more about Amazon Redshift features.

System and architecture overview

An Amazon Redshift data warehouse is an enterprise-class relational database query and management system.

Amazon Redshift supports client connections with many types of applications, including business intelligence (BI), reporting, data, and analytics tools.

When you run analytic queries, you are retrieving, comparing, and evaluating large amounts of data in multiple-stage operations to produce a final result.

Amazon Redshift achieves efficient storage and optimum query performance through a combination of massively parallel processing, columnar data storage, and very efficient, targeted data compression encoding schemes. This section presents an introduction to the Amazon Redshift system architecture.

Topics

- Data warehouse system architecture (p. 3)
 - Performance (p. 5)
 - Columnar storage (p. 7)
 - Workload management (p. 8)
 - Using Amazon Redshift with other services (p. 9)

Data warehouse system architecture

This section introduces the elements of the Amazon Redshift data warehouse architecture as shown in the following figure.



Client applications

Amazon Redshift integrates with various data loading and ETL (extract, transform, and load) tools and business intelligence (BI) reporting, data mining, and analytics tools. Amazon Redshift is based on open standard PostgreSQL, so most existing SQL client applications will work with only minimal changes. For information about important differences between Amazon Redshift SQL and PostgreSQL, see [Amazon Redshift and PostgreSQL \(p. 565\)](#).

Clusters

The core infrastructure component of an Amazon Redshift data warehouse is a *cluster*.

A cluster is composed of one or more *compute nodes*. If a cluster is provisioned with two or more compute nodes, an additional *leader node* coordinates the compute nodes and handles external communication. Your client application interacts directly only with the leader node. The compute nodes are transparent to external applications.

Leader node

The leader node manages communications with client programs and all communication with compute nodes. It parses and develops execution plans to carry out database operations, in particular, the series of steps necessary to obtain results for complex queries. Based on the execution plan, the leader node compiles code, distributes the compiled code to the compute nodes, and assigns a portion of the data to each compute node.

The leader node distributes SQL statements to the compute nodes only when a query references tables that are stored on the compute nodes. All other queries run exclusively on the leader node. Amazon Redshift is designed to implement certain SQL functions only on the leader node. A query that uses any of these functions will return an error if it references tables that reside on the compute nodes. For more information, see [SQL functions supported on the leader node \(p. 564\)](#).

Compute nodes

The leader node compiles code for individual elements of the execution plan and assigns the code to individual compute nodes. The compute nodes run the compiled code and send intermediate results back to the leader node for final aggregation.

Each compute node has its own dedicated CPU and memory, which are determined by the node type. As your workload grows, you can increase the compute capacity of a cluster by increasing the number of nodes, upgrading the node type, or both.

Amazon Redshift provides several node types for your compute needs. For details of each node type, see [Amazon Redshift clusters](#) in the *Amazon Redshift Management Guide*.

Redshift Managed Storage

Data warehouse data is stored in a separate storage tier Redshift Managed Storage (RMS). RMS provides the ability to scale your storage to petabytes using Amazon S3 storage. RMS allows you to scale and pay for compute and storage independently, so that you can size your cluster based only on your compute needs. It automatically uses high-performance SSD-based local storage as tier-1 cache. It also takes advantage of optimizations, such as data block temperature, data block age, and workload patterns to deliver high performance while scaling storage automatically to Amazon S3 when needed without requiring any action.

Node slices

A compute node is partitioned into slices. Each slice is allocated a portion of the node's memory and disk space, where it processes a portion of the workload assigned to the node. The leader node manages distributing data to the slices and apportions the workload for any queries or other database operations to the slices. The slices then work in parallel to complete the operation.

The number of slices per node is determined by the node size of the cluster. For more information about the number of slices for each node size, go to [About clusters and nodes](#) in the *Amazon Redshift Management Guide*.

When you create a table, you can optionally specify one column as the distribution key. When the table is loaded with data, the rows are distributed to the node slices according to the distribution key that is defined for a table. Choosing a good distribution key enables Amazon Redshift to use parallel processing to load data and run queries efficiently. For information about choosing a distribution key, see [Choose the best distribution style \(p. 16\)](#).

Internal network

Amazon Redshift takes advantage of high-bandwidth connections, close proximity, and custom communication protocols to provide private, very high-speed network communication between the leader node and compute nodes. The compute nodes run on a separate, isolated network that client applications never access directly.

Databases

A cluster contains one or more databases. User data is stored on the compute nodes. Your SQL client communicates with the leader node, which in turn coordinates query run with the compute nodes.

Amazon Redshift is a relational database management system (RDBMS), so it is compatible with other RDBMS applications. Although it provides the same functionality as a typical RDBMS, including online transaction processing (OLTP) functions such as inserting and deleting data, Amazon Redshift is optimized for high-performance analysis and reporting of very large datasets.

Amazon Redshift is based on PostgreSQL. Amazon Redshift and PostgreSQL have a number of very important differences that you need to take into account as you design and develop your data warehouse applications. For information about how Amazon Redshift SQL differs from PostgreSQL, see [Amazon Redshift and PostgreSQL \(p. 565\)](#).

Performance

Amazon Redshift achieves extremely fast query run by employing these performance features.

Topics

- [Massively parallel processing \(p. 5\)](#)
- [Columnar data storage \(p. 5\)](#)
- [Data compression \(p. 5\)](#)
- [Query optimizer \(p. 5\)](#)
- [Result caching \(p. 6\)](#)
- [Compiled code \(p. 6\)](#)

Massively parallel processing

Massively parallel processing (MPP) enables fast run of the most complex queries operating on large amounts of data. Multiple compute nodes handle all query processing leading up to final result aggregation, with each core of each node running the same compiled query segments on portions of the entire data.

Amazon Redshift distributes the rows of a table to the compute nodes so that the data can be processed in parallel. By selecting an appropriate distribution key for each table, you can optimize the distribution of data to balance the workload and minimize movement of data from node to node. For more information, see [Choose the best distribution style \(p. 16\)](#).

Loading data from flat files takes advantage of parallel processing by spreading the workload across multiple nodes while simultaneously reading from multiple files. For more information about how to load data into tables, see [Amazon Redshift best practices for loading data \(p. 17\)](#).

Columnar data storage

Columnar storage for database tables drastically reduces the overall disk I/O requirements and is an important factor in optimizing analytic query performance. Storing database table information in a columnar fashion reduces the number of disk I/O requests and reduces the amount of data you need to load from disk. Loading less data into memory enables Amazon Redshift to perform more in-memory processing when executing queries. See [Columnar storage \(p. 7\)](#) for a more detailed explanation.

When columns are sorted appropriately, the query processor is able to rapidly filter out a large subset of data blocks. For more information, see [Choose the best sort key \(p. 15\)](#).

Data compression

Data compression reduces storage requirements, thereby reducing disk I/O, which improves query performance. When you run a query, the compressed data is read into memory, then uncompressed during query run. Loading less data into memory enables Amazon Redshift to allocate more memory to analyzing the data. Because columnar storage stores similar data sequentially, Amazon Redshift is able to apply adaptive compression encodings specifically tied to columnar data types. The best way to enable data compression on table columns is by allowing Amazon Redshift to apply optimal compression encodings when you load the table with data. To learn more about using automatic data compression, see [Loading tables with automatic compression \(p. 87\)](#).

Query optimizer

The Amazon Redshift query run engine incorporates a query optimizer that is MPP-aware and also takes advantage of the columnar-oriented data storage. The Amazon Redshift query optimizer implements

significant enhancements and extensions for processing complex analytic queries that often include multi-table joins, subqueries, and aggregation. To learn more about optimizing queries, see [Tuning query performance \(p. 463\)](#).

Result caching

To reduce query runtime and improve system performance, Amazon Redshift caches the results of certain types of queries in memory on the leader node. When a user submits a query, Amazon Redshift checks the results cache for a valid, cached copy of the query results. If a match is found in the result cache, Amazon Redshift uses the cached results and doesn't run the query. Result caching is transparent to the user.

Result caching is turned on by default. To turn off result caching for the current session, set the [enable_result_cache_for_session \(p. 1691\)](#) parameter to off.

Amazon Redshift uses cached results for a new query when all of the following are true:

- The user submitting the query has access permission to the objects used in the query.
- The table or views in the query haven't been modified.
- The query doesn't use a function that must be evaluated each time it's run, such as GETDATE.
- The query doesn't reference Amazon Redshift Spectrum external tables.
- Configuration parameters that might affect query results are unchanged.
- The query syntactically matches the cached query.

To maximize cache effectiveness and efficient use of resources, Amazon Redshift doesn't cache some large query result sets. Amazon Redshift determines whether to cache query results based on a number of factors. These factors include the number of entries in the cache and the instance type of your Amazon Redshift cluster.

To determine whether a query used the result cache, query the [SVL_QLOG \(p. 1640\)](#) system view. If a query used the result cache, the source_query column returns the query ID of the source query. If result caching wasn't used, the source_query column value is NULL.

The following example shows that queries submitted by userid 104 and userid 102 use the result cache from queries run by userid 100.

```
select userid, query, elapsed, source_query from svl_qlog
where userid > 1
order by query desc;

userid | query | elapsed | source_query
-----+-----+-----+-----
 104 | 629035 |     27 |      628919
 104 | 629034 |     60 |      628900
 104 | 629033 |     23 |      628891
 102 | 629017 | 1229393 |
 102 | 628942 |     28 |      628919
 102 | 628941 |     57 |      628900
 102 | 628940 |     26 |      628891
 100 | 628919 | 84295686 |
 100 | 628900 | 87015637 |
 100 | 628891 | 58808694 |
```

Compiled code

The leader node distributes fully optimized compiled code across all of the nodes of a cluster. Compiling the query decreases the overhead associated with an interpreter and therefore increases the runtime

speed, especially for complex queries. The compiled code is cached and shared across sessions on the same cluster. As a result, future runs of the same query will be faster, often even with different parameters.

The query run engine compiles different code for the JDBC and ODBC connection protocols, so two clients using different protocols each incur the first-time cost of compiling the code. Clients that use the same protocol, however, benefit from sharing the cached code.

Columnar storage

Columnar storage for database tables is an important factor in optimizing analytic query performance, because it drastically reduces the overall disk I/O requirements. It reduces the amount of data you need to load from disk.

The following series of illustrations describe how columnar data storage implements efficiencies, and how that translates into efficiencies when retrieving data into memory.

This first illustration shows how records from database tables are typically stored into disk blocks by row.

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL



Block 1

Block 2

Block 3

In a typical relational database table, each row contains field values for a single record. In row-wise database storage, data blocks store values sequentially for each consecutive column making up the entire row. If block size is smaller than the size of a record, storage for an entire record may take more than one block. If block size is larger than the size of a record, storage for an entire record may take less than one block, resulting in an inefficient use of disk space. In online transaction processing (OLTP) applications, most transactions involve frequently reading and writing all of the values for entire records, typically one record or a small number of records at a time. As a result, row-wise storage is optimal for OLTP databases.

The next illustration shows how with columnar storage, the values for each column are stored sequentially into disk blocks.

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL



Block 1

Using columnar storage, each data block stores values of a single column for multiple rows. As records enter the system, Amazon Redshift transparently converts the data to columnar storage for each of the columns.

In this simplified example, using columnar storage, each data block holds column field values for as many as three times as many records as row-based storage. This means that reading the same number of column field values for the same number of records requires a third of the I/O operations compared to row-wise storage. In practice, using tables with very large numbers of columns and very large row counts, storage efficiency is even greater.

An added advantage is that, since each block holds the same type of data, block data can use a compression scheme selected specifically for the column data type, further reducing disk space and I/O. For more information about compression encodings based on data types, see [Compression encodings \(p. 39\)](#).

The savings in space for storing data on disk also carries over to retrieving and then storing that data in memory. Since many database operations only need to access or operate on one or a small number of columns at a time, you can save memory space by only retrieving blocks for columns you actually need for a query. Where OLTP transactions typically involve most or all of the columns in a row for a small number of records, data warehouse queries commonly read only a few columns for a very large number of rows. This means that reading the same number of column field values for the same number of rows requires a fraction of the I/O operations. It uses a fraction of the memory that would be required for processing row-wise blocks. In practice, using tables with very large numbers of columns and very large row counts, the efficiency gains are proportionally greater. For example, suppose a table contains 100 columns. A query that uses five columns will only need to read about five percent of the data contained in the table. This savings is repeated for possibly billions or even trillions of records for large databases. In contrast, a row-wise database would read the blocks that contain the 95 unneeded columns as well.

Typical database block sizes range from 2 KB to 32 KB. Amazon Redshift uses a block size of 1 MB, which is more efficient and further reduces the number of I/O requests needed to perform any database loading or other operations that are part of query run.

Workload management

Amazon Redshift workload management (WLM) enables users to flexibly manage priorities within workloads so that short, fast-running queries won't get stuck in queues behind long-running queries.

Amazon Redshift WLM creates query queues at runtime according to *service classes*, which define the configuration parameters for various types of queues, including internal system queues and user-accessible queues. From a user perspective, a user-accessible service class and a queue are functionally equivalent. For consistency, this documentation uses the term *queue* to mean a user-accessible service class as well as a runtime queue.

When you run a query, WLM assigns the query to a queue according to the user's user group or by matching a query group that is listed in the queue configuration with a query group label that the user sets at runtime.

Currently, the default for clusters using the default parameter group is to use automatic WLM. Automatic WLM manages query concurrency and memory allocation. For more information, see [Implementing automatic WLM \(p. 493\)](#).

With manual WLM, Amazon Redshift configures one queue with a *concurrency level* of five, which enables up to five queries to run concurrently, plus one predefined Superuser queue, with a concurrency level of one. You can define up to eight queues. Each queue can be configured with a maximum concurrency level of 50. The maximum total concurrency level for all user-defined queues (not including the Superuser queue) is 50.

The easiest way to modify the WLM configuration is by using the Amazon Redshift Management Console. You can also use the Amazon Redshift command line interface (CLI) or the Amazon Redshift API.

For more information about implementing and using workload management, see [Implementing workload management \(p. 491\)](#).

Using Amazon Redshift with other services

Amazon Redshift integrates with other AWS services to enable you to move, transform, and load your data quickly and reliably, using data security features.

Moving data between Amazon Redshift and Amazon S3

Amazon Simple Storage Service (Amazon S3) is a web service that stores data in the cloud. Amazon Redshift leverages parallel processing to read and load data from multiple data files stored in Amazon S3 buckets. For more information, see [Loading data from Amazon S3 \(p. 66\)](#).

You can also use parallel processing to export data from your Amazon Redshift data warehouse to multiple data files on Amazon S3. For more information, see [Unloading data \(p. 144\)](#).

Using Amazon Redshift with Amazon DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service. You can use the COPY command to load an Amazon Redshift table with data from a single Amazon DynamoDB table. For more information, see [Loading data from an Amazon DynamoDB table \(p. 84\)](#).

Importing data from remote hosts over SSH

You can use the COPY command in Amazon Redshift to load data from one or more remote hosts, such as Amazon EMR clusters, Amazon EC2 instances, or other computers. COPY connects to the remote hosts using SSH and runs commands on the remote hosts to generate data. Amazon Redshift supports multiple simultaneous connections. The COPY command reads and loads the output from multiple host sources in parallel. For more information, see [Loading data from remote hosts \(p. 78\)](#).

Automating data loads using AWS Data Pipeline

You can use AWS Data Pipeline to automate data movement and transformation into and out of Amazon Redshift. By using the built-in scheduling capabilities of AWS Data Pipeline, you can schedule and run recurring jobs without having to write your own complex data transfer or transformation logic. For example, you can set up a recurring job to automatically copy data from Amazon DynamoDB into Amazon Redshift. For a tutorial that walks you through the process of creating a pipeline that periodically moves data from Amazon S3 to Amazon Redshift, see [Copy data to Amazon Redshift using AWS Data Pipeline](#) in the AWS Data Pipeline Developer Guide.

Migrating data using AWS Database Migration Service (AWS DMS)

You can migrate data to Amazon Redshift using AWS Database Migration Service. AWS DMS can migrate your data to and from most widely used commercial and open-source databases such as Oracle, PostgreSQL, Microsoft SQL Server, Amazon Redshift, Aurora, DynamoDB, Amazon S3, MariaDB, and MySQL. For more information, see [Using an Amazon Redshift database as a target for AWS Database Migration Service](#).

Amazon Redshift best practices

Following, you can find best practices for planning a proof of concept, designing tables, loading data into tables, and writing queries for Amazon Redshift, and also a discussion of working with Amazon Redshift Advisor.

Amazon Redshift is not the same as other SQL database systems. To fully realize the benefits of the Amazon Redshift architecture, you must specifically design, build, and load your tables to use massively parallel processing, columnar data storage, and columnar data compression. If your data loading and query execution times are longer than you expect, or longer than you want, you might be overlooking key information.

If you are an experienced SQL database developer, we strongly recommend that you review this topic before you begin developing your Amazon Redshift data warehouse.

If you are new to developing SQL databases, this topic is not the best place to start. We recommend that you begin by reading [Getting started using databases](#) and trying the examples yourself.

In this topic, you can find an overview of the most important development principles, along with specific tips, examples, and best practices for implementing those principles. No single practice can apply to every application. Evaluate all of your options before finishing a database design. For more information, see [Working with automatic table optimization \(p. 36\)](#), [Loading data \(p. 63\)](#), [Tuning query performance \(p. 463\)](#), and the reference chapters.

Topics

- [Conducting a proof of concept for Amazon Redshift \(p. 10\)](#)
- [Amazon Redshift best practices for designing tables \(p. 15\)](#)
- [Amazon Redshift best practices for loading data \(p. 17\)](#)
- [Amazon Redshift best practices for designing queries \(p. 21\)](#)
- [Working with recommendations from Amazon Redshift Advisor \(p. 23\)](#)

Conducting a proof of concept for Amazon Redshift

Amazon Redshift is a fast, scalable data warehouse that makes it simple and cost-effective to analyze all your data using standard SQL with your existing business intelligence (BI) tools. Amazon Redshift offers fast performance in a low-cost cloud data warehouse. It uses sophisticated query optimization, accelerated cache, columnar storage on high-performance local disks, and massively parallel query execution.

In the following sections, you can find a framework for building a proof of concept with Amazon Redshift. The framework helps you to use architectural best practices for designing and operating a secure, high-performing, and cost-effective data warehouse. This guidance is based on reviewing designs of thousands of customer architectures across a wide variety of business types and use cases. We have compiled customer experiences to develop this set of best practices to help you develop criteria for evaluating your data warehouse workload.

Overview of the process

Conducting a proof of concept is a three-step process:

1. Identify the goals of the proof of concept – you can work backward from your business requirements and success criteria, and translate them into a technical proof of concept project plan.

2. Set up the proof of concept environment – most of the setup process is a click of few buttons to create your resources. Within minutes, you can have a data warehouse environment ready with data loaded.
3. Complete the proof of concept project plan to ensure that the goals are met.

In the following sections, we go into the details of each step.

Identify the business goals and success criteria

Identifying the goals of the proof of concept plays a critical role in determining what you want to measure as part of the evaluation process. The evaluation criteria should include the current scaling challenges, enhancements to improve your customer's experience of the data warehouse, and methods of addressing your current operational pain points. You can use the following questions to identify the goals of the proof of concept:

- What are your goals for scaling your data warehouse?
- What are the specific service-level agreements whose terms you want to improve?
- What new datasets do you need to include in your data warehouse?
- What are the business-critical SQL queries that you need to test and measure? Make sure to include the full range of SQL complexities, such as the different types of queries (for example, select, insert, update, and delete).
- What are the general types of workloads you plan to test? Examples might include extract-transform-load (ETL) workloads, reporting queries, and batch extracts.

After you have answered these questions, you should be able to establish SMART goals and success criteria for building your proof of concept. For information about setting goals, see [SMART criteria](#) in [Wikipedia](#).

Set up your proof of concept

Because we eliminated hardware provisioning, networking, and software installation from an on-premises data warehouse, trying Amazon Redshift with your own dataset has never been easier. Many of the sizing decisions and estimations that used to be required are now simply a click away. You can flexibly resize your cluster or adjust the ratio of storage versus compute.

Broadly, setting up the Amazon Redshift proof of concept environment is a two-step process. It involves the launching of a data warehouse and then the conversion of the schema and datasets for evaluation.

Choose a starting cluster size

You can choose the node type and number of nodes using the Amazon Redshift console. We recommend that you also test resizing the cluster as part of your proof of concept plan. To get the initial sizing for your cluster, take the following steps:

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Create cluster** to open the configuration page.
3. For **Cluster identifier**, enter a name for your cluster.
4. The following step describes an Amazon Redshift console that is running in an AWS Region that supports RA3 node types. For a list of AWS Regions that support RA3 node types, see [Overview of RA3 node types](#) in the *Amazon Redshift Management Guide*.

If you don't know how large to size your cluster, choose **Help me choose**. Doing this starts a sizing calculator that asks you questions about the size and query characteristics of the data that you plan to

store in your data warehouse. If you know the required size of your cluster (that is, the node type and number of nodes), choose **I'll choose**. Then choose the **Node type** and number of **Nodes** to size your cluster for the proof of concept.

Note

If your organization is eligible and your cluster is being created in an AWS Region where Amazon Redshift Serverless is unavailable, you might be able to create a cluster under the Amazon Redshift free trial program. Choose either **Production** or **Free trial** to answer the question **What are you planning to use this cluster for?** When you choose **Free trial**, you create a configuration with the dc2.large node type. For more information about choosing a free trial, see [Amazon Redshift free trial](#). For a list of AWS Regions where Amazon Redshift Serverless is available, see the endpoints listed for the [Redshift Serverless API](#) in the [Amazon Web Services General Reference](#).

5. After you enter all required cluster properties, choose **Create cluster** to launch your data warehouse.

For more details about creating clusters with the Amazon Redshift console, see [Creating a cluster](#) in the [Amazon Redshift Management Guide](#).

Convert the schema and set up the datasets for the proof of concept

If you don't have an existing data warehouse, skip this section and see [Amazon Redshift Getting Started Guide](#). *Amazon Redshift Getting Started Guide* provides a tutorial to create a cluster and examples of setting up data in Amazon Redshift.

When migrating from your existing data warehouse, you can convert schema, code, and data using the AWS Schema Conversion Tool and the AWS Database Migration Service. Your choice of tools depends on the source of your data and optional ongoing replications. For more information, see [What Is the AWS Schema Conversion Tool?](#) in the *AWS Schema Conversion Tool User Guide* and [What Is AWS Database Migration Service?](#) in the *AWS Database Migration Service User Guide*. The following can help you set up your data in Amazon Redshift:

- [Migrate Your Data Warehouse to Amazon Redshift Using the AWS Schema Conversion Tool](#) – this blog post provides an overview on how you can use the AWS SCT data extractors to migrate your existing data warehouse to Amazon Redshift. The AWS SCT tool can migrate your data from many legacy platforms (such as Oracle, Greenplum, Netezza, Teradata, Microsoft SQL Server, or Vertica).
- Optionally, you can also use the AWS Database Migration Service for ongoing replications of changed data from the source. For more information, see [Using an Amazon Redshift Database as a Target for AWS Database Migration Service](#) in the *AWS Database Migration Service User Guide*.

Amazon Redshift is a relational database management system (RDBMS). As such, it can run many types of data models including star schemas, snowflake schemas, data vault models, and simple, flat, or normalized tables. After setting up your schemas in Amazon Redshift, you can take advantage of massively parallel processing and columnar data storage for fast analytical queries out of the box. For information about types of schemas, see [star schema](#), [snowflake schema](#), and [data vault modeling](#) in [Wikipedia](#).

Checklist for a complete evaluation

Make sure that a complete evaluation meets all your data warehouse needs. Consider including the following items in your success criteria:

- **Data load time** – using the COPY command is a common way to test how long it takes to load data. For more information, see [Amazon Redshift best practices for loading data \(p. 17\)](#).

- **Throughput of the cluster** – measuring queries per hour is a common way to determine throughput. To do so, set up a test to run typical queries for your workload.
- **Data security** – you can easily encrypt data at rest and in transit with Amazon Redshift. You also have a number of options for managing keys. Amazon Redshift also supports single sign-on integration. Amazon Redshift pricing includes built-in security, data compression, backup storage, and data transfer.
- **Third-party tools integration** – you can use either a JDBC or ODBC connection to integrate with business intelligence and other external tools.
- **Interoperability with other AWS services** – Amazon Redshift integrates with other AWS services, such as Amazon EMR, Amazon QuickSight, AWS Glue, Amazon S3, and Amazon Kinesis. You can use this integration when setting up and managing your data warehouse.
- **Backups and snapshots** – backups and snapshots are created automatically. You can also create a point-in-time snapshot at any time or on a schedule. Try using a snapshot and creating a second cluster as part of your evaluation. Evaluate if your development and testing organizations can use the cluster.
- **Resizing** – your evaluation should include increasing the number or types of Amazon Redshift nodes. Evaluate that the workload throughput before and after a resize meets any variability of the volume of your workload. For more information, see [Resizing clusters in Amazon Redshift](#) in the *Amazon Redshift Management Guide*.
- **Concurrency scaling** – this feature helps you handle variability of traffic volume in your data warehouse. With concurrency scaling, you can support virtually unlimited concurrent users and concurrent queries, with consistently fast query performance. For more information, see [Working with concurrency scaling \(p. 515\)](#).
- **Automatic workload management (WLM)** – prioritize your business critical queries over other queries by using automatic WLM. Try setting up queues based on your workloads (for example, a queue for ETL and a queue for reporting). Then enable automatic WLM to allocate the concurrency and memory resources dynamically. For more information, see [Implementing automatic WLM \(p. 493\)](#).
- **Amazon Redshift Advisor** – the Advisor develops customized recommendations to increase performance and optimize costs by analyzing your workload and usage metrics for your cluster. Sign in to the Amazon Redshift console to view Advisor recommendations. For more information, see [Working with recommendations from Amazon Redshift Advisor \(p. 23\)](#).
- **Table design** – Amazon Redshift provides great performance out of the box for most workloads. When you create a table, the default sort key and distribution key is AUTO. For more information, see [Working with automatic table optimization \(p. 36\)](#).
- **Support** – we strongly recommend that you evaluate AWS Support as part of your evaluation. Also, make sure to talk to your account manager about your proof of concept. AWS can help with technical guidance and credits for the proof of concept if you qualify. If you don't find the help you're looking for, you can talk directly to the Amazon Redshift team. For help, submit the form at [Request support for your Amazon Redshift proof-of-concept](#).
- **Lake house integration** – with built-in integration, try using the out-of-box Amazon Redshift Spectrum feature. With Redshift Spectrum, you can extend the data warehouse into your data lake and run queries against petabytes of data in Amazon S3 using your existing cluster. For more information, see [Querying external data using Amazon Redshift Spectrum \(p. 248\)](#).

Develop a project plan for your evaluation

Some of the following techniques for creating query benchmarks might help support your Amazon Redshift evaluation:

- Assemble a list of queries for each runtime category. Having a sufficient number (for example, 30 per category) helps ensure that your evaluation reflects a real-world data warehouse implementation. Add a unique identifier to associate each query that you include in your evaluation with one of the

categories you establish for your evaluation. You can then use these unique identifiers to determine throughput from the system tables.

You can also create a query group to organize your evaluation queries. For example, if you have established a "Reporting" category for your evaluation, you might create a coding system to tag your evaluation queries with the word "Report." You can then identify individual queries within reporting as R1, R2, and so on. The following example demonstrates this approach.

```
SELECT 'Reporting' AS query_category, 'R1' as query_id, * FROM customers;
```

```
SELECT query, datediff(seconds, starttime, endtime)
  FROM stl_query
 WHERE
    querytxt LIKE '%Reporting%'
    and starttime >= '2018-04-15 00:00'
    and endtime < '2018-04-15 23:59';
```

When you have associated a query with an evaluation category, you can use a unique identifier to determine throughput from the system tables for each category.

- Test throughput with historical user or ETL queries that have a variety of runtimes in your existing data warehouse. You might use a load testing utility, such as the open-source [JMeter](#) or a custom utility. If so, make sure that your utility does the following:
 - It can take the network transmission time into account.
 - It evaluates execution time based on throughput of the internal system tables. For information about how to do this, see [Analyzing the query summary \(p. 476\)](#).
- Identify all the various permutations that you plan to test during your evaluation. The following list provides some common variables:
 - Cluster size
 - Node type
 - Load testing duration
 - Concurrency settings
- Reduce the cost of your proof of concept by pausing your cluster during off-hours and weekends. When a cluster is paused, on-demand compute billing is suspended. To run tests on the cluster, resume per-second billing. You can also create a schedule to pause and resume your cluster automatically. For more information, see [Pausing and resuming clusters](#) in the *Amazon Redshift Management Guide*.

At this stage, you're ready to complete your project plan and evaluate results.

Additional resources to help your evaluation

To help your Amazon Redshift evaluation, see the following:

- [Service highlights and pricing](#) – this product detail page provides the Amazon Redshift value proposition, service highlights, and pricing.
- [Amazon Redshift Getting Started Guide](#) – this guide provides a tutorial of using Amazon Redshift to create a sample cluster and work with sample data.
- [Getting started with Amazon Redshift Spectrum \(p. 250\)](#) – in this tutorial, you learn how to use Redshift Spectrum to query data directly from files on Amazon S3.
- [Amazon Redshift management overview](#) – this topic in the *Amazon Redshift Management Guide* provides an overview of Amazon Redshift.

- Optimize Amazon Redshift for performance with BI tools – consider integration with tools such as [Tableau](#), [Power BI](#), and others.
- [Amazon Redshift Advisor recommendations \(p. 24\)](#) – contains explanations and details for each Advisor recommendation.
- [What's new in Amazon Redshift](#) – announcements that help you keep track of new features and enhancements.
- [Improved speed and scalability](#) – this blog post summarizes recent Amazon Redshift improvements.

Need help?

Make sure to talk to your account manager to let them know about your proof of concept. AWS can help with technical guidance and credits for the proof of concept if you qualify. If you don't find the help you are looking for, you can talk directly to the Amazon Redshift team. For help, submit the form at [Request support for your Amazon Redshift proof-of-concept](#).

Amazon Redshift best practices for designing tables

As you plan your database, certain key table design decisions heavily influence overall query performance. These design choices also have a significant effect on storage requirements, which in turn affects query performance by reducing the number of I/O operations and minimizing the memory required to process queries.

In this section, you can find a summary of the most important design decisions and best practices for optimizing query performance. [Working with automatic table optimization \(p. 36\)](#) provides more detailed explanations and examples of table design options.

Topics

- [Choose the best sort key \(p. 15\)](#)
- [Choose the best distribution style \(p. 16\)](#)
- [Let COPY choose compression encodings \(p. 17\)](#)
- [Define primary key and foreign key constraints \(p. 17\)](#)
- [Use date/time data types for date columns \(p. 17\)](#)

Choose the best sort key

Amazon Redshift stores your data on disk in sorted order according to the sort key. The Amazon Redshift query optimizer uses sort order when it determines optimal query plans.

Note

When you use automatic table optimization, you don't need to choose the sort key of your table. For more information, see [Working with automatic table optimization \(p. 36\)](#).

Some suggestions for the best approach follow:

- To have Amazon Redshift choose the appropriate sort order, specify AUTO for the sort key.
- If recent data is queried most frequently, specify the timestamp column as the leading column for the sort key.

Queries are more efficient because they can skip entire blocks that fall outside the time range.

- If you do frequent range filtering or equality filtering on one column, specify that column as the sort key.

Amazon Redshift can skip reading entire blocks of data for that column. It can do so because it tracks the minimum and maximum column values stored on each block and can skip blocks that don't apply to the predicate range.

- If you frequently join a table, specify the join column as both the sort key and the distribution key.

Doing this enables the query optimizer to choose a sort merge join instead of a slower hash join. Because the data is already sorted on the join key, the query optimizer can bypass the sort phase of the sort merge join.

Choose the best distribution style

When you run a query, the query optimizer redistributes the rows to the compute nodes as needed to perform any joins and aggregations. The goal in selecting a table distribution style is to minimize the impact of the redistribution step by locating the data where it needs to be before the query is run.

Note

When you use automatic table optimization, you don't need to choose the distribution style of your table. For more information, see [Working with automatic table optimization \(p. 36\)](#).

Some suggestions for the best approach follow:

1. Distribute the fact table and one dimension table on their common columns.

Your fact table can have only one distribution key. Any tables that join on another key aren't collocated with the fact table. Choose one dimension to colocate based on how frequently it is joined and the size of the joining rows. Designate both the dimension table's primary key and the fact table's corresponding foreign key as the DISTKEY.

2. Choose the largest dimension based on the size of the filtered dataset.

Only the rows that are used in the join must be distributed, so consider the size of the dataset after filtering, not the size of the table.

3. Choose a column with high cardinality in the filtered result set.

If you distribute a sales table on a date column, for example, you should probably get fairly even data distribution, unless most of your sales are seasonal. However, if you commonly use a range-restricted predicate to filter for a narrow date period, most of the filtered rows occur on a limited set of slices and the query workload is skewed.

4. Change some dimension tables to use ALL distribution.

If a dimension table cannot be colocated with the fact table or other important joining tables, you can improve query performance significantly by distributing the entire table to all of the nodes. Using ALL distribution multiplies storage space requirements and increases load times and maintenance operations, so you should weigh all factors before choosing ALL distribution.

To have Amazon Redshift choose the appropriate distribution style, specify AUTO for the distribution style.

For more information about choosing distribution styles, see [Working with data distribution styles \(p. 49\)](#).

Let COPY choose compression encodings

You can specify compression encodings when you create a table, but in most cases, automatic compression produces the best results.

ENCODE AUTO is the default for tables. When a table is set to ENCODE AUTO, Amazon Redshift automatically manages compression encoding for all columns in the table. For more information, see [CREATE TABLE \(p. 823\)](#) and [ALTER TABLE \(p. 640\)](#).

The COPY command analyzes your data and applies compression encodings to an empty table automatically as part of the load operation.

Automatic compression balances overall performance when choosing compression encodings. Range-restricted scans might perform poorly if sort key columns are compressed much more highly than other columns in the same query. As a result, automatic compression chooses a less efficient compression encoding to keep the sort key columns balanced with other columns.

Suppose that your table's sort key is a date or timestamp and the table uses many large varchar columns. In this case, you might get better performance by not compressing the sort key column at all. Run the [ANALYZE COMPRESSION \(p. 665\)](#) command on the table, then use the encodings to create a new table, but leave out the compression encoding for the sort key.

There is a performance cost for automatic compression encoding, but only if the table is empty and does not already have compression encoding. For short-lived tables and tables that you create frequently, such as staging tables, load the table once with automatic compression or run the ANALYZE COMPRESSION command. Then use those encodings to create new tables. You can add the encodings to the CREATE TABLE statement, or use CREATE TABLE LIKE to create a new table with the same encoding.

For more information, see [Loading tables with automatic compression \(p. 87\)](#).

Define primary key and foreign key constraints

Define primary key and foreign key constraints between tables wherever appropriate. Even though they are informational only, the query optimizer uses those constraints to generate more efficient query plans.

Do not define primary key and foreign key constraints unless your application enforces the constraints. Amazon Redshift does not enforce unique, primary-key, and foreign-key constraints.

See [Defining table constraints \(p. 62\)](#) for additional information about how Amazon Redshift uses constraints.

Use date/time data types for date columns

Amazon Redshift stores DATE and TIMESTAMP data more efficiently than CHAR or VARCHAR, which results in better query performance. Use the DATE or TIMESTAMP data type, depending on the resolution you need, rather than a character type when storing date/time information. For more information, see [Datetime types \(p. 585\)](#).

Amazon Redshift best practices for loading data

Topics

- [Take the loading data tutorial \(p. 18\)](#)
- [Use a COPY command to load data \(p. 18\)](#)

- [Use a single COPY command to load from multiple files \(p. 18\)](#)
- [Loading data files \(p. 18\)](#)
- [Compressing your data files \(p. 19\)](#)
- [Verify data files before and after a load \(p. 19\)](#)
- [Use a multi-row insert \(p. 19\)](#)
- [Use a bulk insert \(p. 20\)](#)
- [Load data in sort key order \(p. 20\)](#)
- [Load data in sequential blocks \(p. 20\)](#)
- [Use time-series tables \(p. 21\)](#)
- [Use a staging table to perform a merge \(upsert\) \(p. 21\)](#)
- [Schedule around maintenance windows \(p. 21\)](#)

Loading very large datasets can take a long time and consume a lot of computing resources. How your data is loaded can also affect query performance. This section presents best practices for loading data efficiently using COPY commands, bulk inserts, and staging tables.

Take the loading data tutorial

[Tutorial: Loading data from Amazon S3 \(p. 124\)](#) walks you beginning to end through the steps to upload data to an Amazon S3 bucket and then use the COPY command to load the data into your tables. The tutorial includes help with troubleshooting load errors and compares the performance difference between loading from a single file and loading from multiple files.

Use a COPY command to load data

The COPY command loads data in parallel from Amazon S3, Amazon EMR, Amazon DynamoDB, or multiple data sources on remote hosts. COPY loads large amounts of data much more efficiently than using INSERT statements, and stores the data more effectively as well.

For more information about using the COPY command, see [Loading data from Amazon S3 \(p. 66\)](#) and [Loading data from an Amazon DynamoDB table \(p. 84\)](#).

Use a single COPY command to load from multiple files

Amazon Redshift can automatically load in parallel from multiple compressed data files.

However, if you use multiple concurrent COPY commands to load one table from multiple files, Amazon Redshift is forced to perform a serialized load. This type of load is much slower and requires a VACUUM process at the end if the table has a sort column defined. For more information about using COPY to load data in parallel, see [Loading data from Amazon S3 \(p. 66\)](#).

Loading data files

Source-data files come in different formats and use varying compression algorithms. When loading data with the COPY command, Amazon Redshift loads all of the files referenced by the Amazon S3 bucket prefix. (The prefix is a string of characters at the beginning of the object key name.) If the prefix refers to multiple files or files that can be split, Amazon Redshift loads the data in parallel, taking advantage of Amazon Redshift's MPP architecture. This divides the workload among the nodes in the cluster. In contrast, when you load data from a file that can't be split, Amazon Redshift is forced to perform a

serialized load, which is much slower. The following sections describe the recommended way to load different file types into Amazon Redshift, depending on their format and compression.

Loading data from files that can be split

The following files can be automatically split when their data is loaded:

- an uncompressed CSV file
- a CSV file compressed with BZIP
- a columnar file (Parquet/ORC)

Amazon Redshift automatically splits files 128MB or larger into chunks. Columnar files, specifically Parquet and ORC, aren't split if they're less than 128MB. Redshift makes use of slices working in parallel to load the data. This provides fast load performance.

Loading data from files that can't be split

File types such as JSON, or CSV, when compressed with other compression algorithms, such as GZIP, aren't automatically split. For these we recommend manually splitting the data into multiple smaller files that are close in size, from 1 MB to 1 GB after compression. Additionally, make the number of files a multiple of the number of slices in your cluster. For more information about how to split your data into multiple files and examples of loading data using COPY, see [Loading data from Amazon S3](#).

Compressing your data files

When you want to compress large load files, we recommend that you use gzip, lzop, bzip2, or zstandard to compress them and split the data into multiple smaller files.

Specify the GZIP, LZOP, BZIP2, or ZSTD option with the COPY command. This example loads the TIME table from a pipe-delimited lzop file.

```
copy time
from 's3://mybucket/data/timerows.lzo'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
lzop
delimiter '|';
```

There are instances when you don't have to split uncompressed data files. For more information about splitting your data and examples of using COPY to load data, see [Loading data from Amazon S3 \(p. 66\)](#).

Verify data files before and after a load

When you load data from Amazon S3, first upload your files to your Amazon S3 bucket, then verify that the bucket contains all the correct files, and only those files. For more information, see [Verifying that the correct files are present in your bucket \(p. 69\)](#).

After the load operation is complete, query the [STL_LOAD_COMMITS \(p. 1523\)](#) system table to verify that the expected files were loaded. For more information, see [Verifying that the data loaded correctly \(p. 86\)](#).

Use a multi-row insert

If a COPY command is not an option and you require SQL inserts, use a multi-row insert whenever possible. Data compression is inefficient when you add data only one row or a few rows at a time.

Multi-row inserts improve performance by batching up a series of inserts. The following example inserts three rows into a four-column table using a single INSERT statement. This is still a small insert, shown simply to illustrate the syntax of a multi-row insert.

```
insert into category_stage values
(default, default, default, default),
(20, default, 'Country', default),
(21, 'Concerts', 'Rock', default);
```

For more details and examples, see [INSERT \(p. 902\)](#).

Use a bulk insert

Use a bulk insert operation with a SELECT clause for high-performance data insertion.

Use the [INSERT \(p. 902\)](#) and [CREATE TABLE AS \(p. 838\)](#) commands when you need to move data or a subset of data from one table into another.

For example, the following INSERT statement selects all of the rows from the CATEGORY table and inserts them into the CATEGORY_STAGE table.

```
insert into category_stage
(select * from category);
```

The following example creates CATEGORY_STAGE as a copy of CATEGORY and inserts all of the rows in CATEGORY into CATEGORY_STAGE.

```
create table category_stage as
select * from category;
```

Load data in sort key order

Load your data in sort key order to avoid needing to vacuum.

If each batch of new data follows the existing rows in your table, your data is properly stored in sort order, and you don't need to run a vacuum. You don't need to presort the rows in each load because COPY sorts each batch of incoming data as it loads.

For example, suppose that you load data every day based on the current day's activity. If your sort key is a timestamp column, your data is stored in sort order. This order occurs because the current day's data is always appended at the end of the previous day's data. For more information, see [Loading your data in sort key order \(p. 117\)](#). For more information about vacuum operations, see [Vacuuming tables](#).

Load data in sequential blocks

If you need to add a large quantity of data, load the data in sequential blocks according to sort order to eliminate the need to vacuum.

For example, suppose that you need to load a table with events from January 2017 to December 2017. Assuming each month is in a single file, load the rows for January, then February, and so on. Your table is completely sorted when your load completes, and you don't need to run a vacuum. For more information, see [Use time-series tables \(p. 21\)](#).

When loading very large datasets, the space required to sort might exceed the total available space. By loading data in smaller blocks, you use much less intermediate sort space during each load. In addition, loading smaller blocks make it easier to restart if the COPY fails and is rolled back.

Use time-series tables

If your data has a fixed retention period, you can organize your data as a sequence of time-series tables. In such a sequence, each table is identical but contains data for different time ranges.

You can easily remove old data simply by running a DROP TABLE command on the corresponding tables. This approach is much faster than running a large-scale DELETE process and saves you from having to run a subsequent VACUUM process to reclaim space. To hide the fact that the data is stored in different tables, you can create a UNION ALL view. When you delete old data, refine your UNION ALL view to remove the dropped tables. Similarly, as you load new time periods into new tables, add the new tables to the view. To signal the optimizer to skip the scan on tables that don't match the query filter, your view definition filters for the date range that corresponds to each table.

Avoid having too many tables in the UNION ALL view. Each additional table adds a small processing time to the query. Tables don't need to use the same time frame. For example, you might have tables for differing time periods, such as daily, monthly, and yearly.

If you use time-series tables with a timestamp column for the sort key, you effectively load your data in sort key order. Doing this eliminates the need to vacuum to re-sort the data. For more information, see [Loading your data in sort key order \(p. 117\)](#).

Use a staging table to perform a merge (upsert)

You can efficiently update and insert new data by loading your data into a staging table first.

Amazon Redshift doesn't support a single *merge* statement (update or insert, also known as an *upsert*) to insert and update data from a single data source. However, you can effectively perform a merge operation. To do so, load your data into a staging table and then join the staging table with your target table for an UPDATE statement and an INSERT statement. For instructions, see [Updating and inserting new data \(p. 96\)](#).

Schedule around maintenance windows

If a scheduled maintenance occurs while a query is running, the query is terminated and rolled back and you need to restart it. Schedule long-running operations, such as large data loads or VACUUM operation, to avoid maintenance windows. You can also minimize the risk, and make restarts easier when they are needed, by performing data loads in smaller increments and managing the size of your VACUUM operations. For more information, see [Load data in sequential blocks \(p. 20\)](#) and [Vacuuming tables \(p. 110\)](#).

Amazon Redshift best practices for designing queries

To maximize query performance, follow these recommendations when creating queries:

- Design tables according to best practices to provide a solid foundation for query performance. For more information, see [Amazon Redshift best practices for designing tables \(p. 15\)](#).
- Avoid using select *. Include only the columns you specifically need.
- Use a [CASE conditional expression \(p. 1056\)](#) to perform complex aggregations instead of selecting from the same table multiple times.
- Don't use cross-joins unless absolutely necessary. These joins without a join condition result in the Cartesian product of two tables. Cross-joins are typically run as nested-loop joins, which are the slowest of the possible join types.

- Use subqueries in cases where one table in the query is used only for predicate conditions and the subquery returns a small number of rows (less than about 200). The following example uses a subquery to avoid joining the LISTING table.

```
select sum(sales.qtysold)
from sales
where salesid in (select listid from listing where listtime > '2008-12-26');
```

- Use predicates to restrict the dataset as much as possible.
- In the predicate, use the least expensive operators that you can. [Comparison condition \(p. 605\)](#) operators are preferable to [LIKE \(p. 611\)](#) operators. LIKE operators are still preferable to [SIMILAR TO \(p. 614\)](#) or [POSIX operators \(p. 616\)](#).
- Avoid using functions in query predicates. Using them can drive up the cost of the query by requiring large numbers of rows to resolve the intermediate steps of the query.
- If possible, use a WHERE clause to restrict the dataset. The query planner can then use row order to help determine which records match the criteria, so it can skip scanning large numbers of disk blocks. Without this, the query execution engine must scan participating columns entirely.
- Add predicates to filter tables that participate in joins, even if the predicates apply the same filters. The query returns the same result set, but Amazon Redshift is able to filter the join tables before the scan step and can then efficiently skip scanning blocks from those tables. Redundant filters aren't needed if you filter on a column that's used in the join condition.

For example, suppose that you want to join SALES and LISTING to find ticket sales for tickets listed after December, grouped by seller. Both tables are sorted by date. The following query joins the tables on their common key and filters for listing.listtime values greater than December 1.

```
select listing.sellerid, sum(sales.qtysold)
from sales, listing
where sales.salesid = listing.listid
and listing.listtime > '2008-12-01'
group by 1 order by 1;
```

The WHERE clause doesn't include a predicate for sales.saletime, so the execution engine is forced to scan the entire SALES table. If you know the filter would result in fewer rows participating in the join, then add that filter as well. The following example cuts execution time significantly.

```
select listing.sellerid, sum(sales.qtysold)
from sales, listing
where sales.salesid = listing.listid
and listing.listtime > '2008-12-01'
and sales.saletime > '2008-12-01'
group by 1 order by 1;
```

- Use sort keys in the GROUP BY clause so the query planner can use more efficient aggregation. A query might qualify for one-phase aggregation when its GROUP BY list contains only sort key columns, one of which is also the distribution key. The sort key columns in the GROUP BY list must include the first sort key, then other sort keys that you want to use in sort key order. For example, it is valid to use the first sort key, the first and second sort keys, the first, second, and third sort keys, and so on. It is not valid to use the first and third sort keys.

You can confirm the use of one-phase aggregation by running the [EXPLAIN \(p. 881\)](#) command and looking for XN_GroupAggregate in the aggregation step of the query.

- If you use both GROUP BY and ORDER BY clauses, make sure that you put the columns in the same order in both. That is, use the approach just following.

```
group by a, b, c
```

```
order by a, b, c
```

Don't use the following approach.

```
group by b, c, a  
order by a, b, c
```

Working with recommendations from Amazon Redshift Advisor

To help you improve the performance and decrease the operating costs for your Amazon Redshift cluster, Amazon Redshift Advisor offers you specific recommendations about changes to make. Advisor develops its customized recommendations by analyzing performance and usage metrics for your cluster. These tailored recommendations relate to operations and cluster settings. To help you prioritize your optimizations, Advisor ranks recommendations by order of impact.

Advisor bases its recommendations on observations regarding performance statistics or operations data. Advisor develops observations by running tests on your clusters to determine if a test value is within a specified range. If the test result is outside of that range, Advisor generates an observation for your cluster. At the same time, Advisor creates a recommendation about how to bring the observed value back into the best-practice range. Advisor only displays recommendations that should have a significant impact on performance and operations. When Advisor determines that a recommendation has been addressed, it removes it from your recommendation list.

For example, suppose that your data warehouse contains a large number of uncompressed table columns. In this case, you can save on cluster storage costs by rebuilding tables using the ENCODE parameter to specify column compression. In another example, suppose that Advisor observes that your cluster contains a significant amount of data in uncompressed table data. In this case, it provides you with the SQL code block to find the table columns that are candidates for compression and resources that describe how to compress those columns.

Amazon Redshift Regions

The Amazon Redshift Advisor feature is available only in the following AWS Regions:

- US East (N. Virginia) Region (us-east-1)
- US East (Ohio) Region (us-east-2)
- US West (N. California) Region (us-west-1)
- US West (Oregon) Region (us-west-2)
- Asia Pacific (Hong Kong) Region (ap-east-1)
- Asia Pacific (Mumbai) Region (ap-south-1)
- Asia Pacific (Seoul) Region (ap-northeast-2)
- Asia Pacific (Singapore) Region (ap-southeast-1)
- Asia Pacific (Sydney) Region (ap-southeast-2)
- Asia Pacific (Tokyo) Region (ap-northeast-1)
- Canada (Central) Region (ca-central-1)
- China (Beijing) Region (cn-north-1)
- China (Ningxia) Region (cn-northwest-1)

- Europe (Frankfurt) Region (eu-central-1)
- Europe (Ireland) Region (eu-west-1)
- Europe (London) Region (eu-west-2)
- Europe (Paris) Region (eu-west-3)
- Europe (Stockholm) Region (eu-north-1)
- Middle East (Bahrain) Region (me-south-1)
- South America (São Paulo) Region (sa-east-1)

Topics

- [Viewing Amazon Redshift Advisor recommendations on the console \(p. 24\)](#)
- [Amazon Redshift Advisor recommendations \(p. 24\)](#)

Viewing Amazon Redshift Advisor recommendations on the console

You can view Amazon Redshift Advisor analysis results and recommendations on the AWS Management Console.

To view Amazon Redshift Advisor recommendations on the console

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Advisor**.
3. Expand each recommendation to see more details. On this page, you can sort and group recommendations.

Amazon Redshift Advisor recommendations

Amazon Redshift Advisor offers recommendations about how to optimize your Amazon Redshift cluster to increase performance and save on operating costs. You can find explanations for each recommendation in the console, as described preceding. You can find further details on these recommendations in the following sections.

Topics

- [Compress Amazon S3 file objects loaded by COPY \(p. 25\)](#)
- [Isolate multiple active databases \(p. 25\)](#)
- [Reallocate workload management \(WLM\) memory \(p. 26\)](#)
- [Skip compression analysis during COPY \(p. 27\)](#)
- [Split Amazon S3 objects loaded by COPY \(p. 28\)](#)
- [Update table statistics \(p. 29\)](#)
- [Enable short query acceleration \(p. 31\)](#)
- [Alter distribution keys on tables \(p. 31\)](#)
- [Alter sort keys on tables \(p. 32\)](#)
- [Alter compression encodings on columns \(p. 33\)](#)
- [Data type recommendations \(p. 33\)](#)

Compress Amazon S3 file objects loaded by COPY

The COPY command takes advantage of the massively parallel processing (MPP) architecture in Amazon Redshift to read and load data in parallel. It can read files from Amazon S3, DynamoDB tables, and text output from one or more remote hosts.

When loading large amounts of data, we strongly recommend using the COPY command to load compressed data files from S3. Compressing large datasets saves time uploading the files to Amazon S3. COPY can also speed up the load process by uncompressing the files as they are read.

Analysis

Long-running COPY commands that load large uncompressed datasets often have an opportunity for considerable performance improvement. The Advisor analysis identifies COPY commands that load large uncompressed datasets. In such a case, Advisor generates a recommendation to implement compression on the source files in Amazon S3.

Recommendation

Ensure that each COPY that loads a significant amount of data, or runs for a significant duration, ingests compressed data objects from Amazon S3. You can identify the COPY commands that load large uncompressed datasets from Amazon S3 by running the following SQL command as a superuser.

```
SELECT
    wq.userid, query, exec_start_time AS starttime, COUNT(*) num_files,
    ROUND(MAX(wq.total_exec_time/1000000.0),2) execution_secs,
    ROUND(SUM(transfer_size)/(1024.0*1024.0),2) total_mb,
    SUBSTRING(querytxt,1,60) copy_sql
FROM stl_s3client s
JOIN stl_query q USING (query)
JOIN stl_wlm_query wq USING (query)
WHERE s.userid>1 AND http_method = 'GET'
    AND POSITION('COPY ANALYZE' IN querytxt) = 0
    AND aborted = 0 AND final_state='Completed'
GROUP BY 1, 2, 3, 7
HAVING SUM(transfer_size) = SUM(data_size)
AND SUM(transfer_size)/(1024*1024) >= 5
ORDER BY 6 DESC, 5 DESC;
```

If the staged data remains in Amazon S3 after you load it, which is common in data lake architectures, storing this data in a compressed form can reduce your storage costs.

Implementation tips

- The ideal object size is 1–128 MB after compression.
- You can compress files with gzip, lzop, or bzip2 format.

Isolate multiple active databases

As a best practice, we recommend isolating databases in Amazon Redshift from one another. Queries run in a specific database and can't access data from any other database on the cluster. However, the queries that you run in all databases of a cluster share the same underlying cluster storage space and compute resources. When a single cluster contains multiple active databases, their workloads are usually unrelated.

Analysis

The Advisor analysis reviews all databases on the cluster for active workloads running at the same time. If there are active workloads running at the same time, Advisor generates a recommendation to consider migrating databases to separate Amazon Redshift clusters.

Recommendation

Consider moving each actively queried database to a separate dedicated cluster. Using a separate cluster can reduce resource contention and improve query performance. It can do so because it allows you to set the size for each cluster for the storage, cost, and performance needs of each workload. Also, unrelated workloads often benefit from different workload management configurations.

To identify which databases are actively used, you can run this SQL command as a superuser.

```
SELECT database,
       COUNT(*) as num_queries,
       AVG(DATEDIFF(sec,starttime,endtime)) avg_duration,
       MIN(starttime) as oldest_ts,
       MAX(endtime) as latest_ts
  FROM stl_query
 WHERE userid > 1
 GROUP BY database;
```

Implementation tips

- Because a user must connect to each database specifically, and queries can only access a single database, moving databases to separate clusters has minimal impact for users.
- One option to move a database is to take the following steps:
 1. Temporarily restore a snapshot of the current cluster to a cluster of the same size.
 2. Delete all databases from the new cluster except the target database to be moved.
 3. Resize the cluster to an appropriate node type and count for the database's workload.

Reallocate workload management (WLM) memory

Amazon Redshift routes user queries to [Implementing manual WLM \(p. 499\)](#) for processing. Workload management (WLM) defines how those queries are routed to the queues. Amazon Redshift allocates each queue a portion of the cluster's available memory. A queue's memory is divided among the queue's query slots.

When a queue is configured with more slots than the workload requires, the memory allocated to these unused slots goes underutilized. Reducing the configured slots to match the peak workload requirements redistributes the underutilized memory to active slots, and can result in improved query performance.

Analysis

The Advisor analysis reviews workload concurrency requirements to identify query queues with unused slots. Advisor generates a recommendation to reduce the number of slots in a queue when it finds the following:

- A queue with slots that are completely inactive throughout the analysis.
- A queue with more than four slots that had at least two inactive slots throughout the analysis.

Recommendation

Reducing the configured slots to match peak workload requirements redistributes underutilized memory to active slots. Consider reducing the configured slot count for queues where the slots have never been

fully used. To identify these queues, you can compare the peak hourly slot requirements for each queue by running the following SQL command as a superuser.

```

WITH
    generate_dt_series AS (select sysdate - (n * interval '5 second') as dt from (select
        row_number() over () as n from stl_scan limit 17280)),
    apex AS (
        SELECT iq.dt, iq.service_class, iq.num_query_tasks, count(iq.slot_count) as
        service_class_queries, sum(iq.slot_count) as service_class_slots
        FROM
            (select gds.dt, wq.service_class, wscc.num_query_tasks, wq.slot_count
            FROM stl_wlm_query wq
            JOIN stv_wlm_service_class_config wscc ON (wscc.service_class = wq.service_class
            AND wscc.service_class > 5)
            JOIN generate_dt_series gds ON (wq.service_class_start_time <= gds.dt AND
            wq.service_class_end_time > gds.dt)
            WHERE wq.userid > 1 AND wq.service_class > 5) iq
            GROUP BY iq.dt, iq.service_class, iq.num_query_tasks),
        maxes as (SELECT apex.service_class, trunc(apex.dt) as d, date_part(h,apex.dt) as
        dt_h, max(service_class_slots) max_service_class_slots
            from apex group by apex.service_class, apex.dt, date_part(h,apex.dt))
    SELECT apex.service_class - 5 AS queue, apex.service_class, apex.num_query_tasks AS
    max_wlm_concurrency, maxes.d AS day, maxes.dt_h || ':00' - ' || maxes.dt_h || ':59' as
    hour, MAX(apex.service_class_slots) as max_service_class_slots
    FROM apex
    JOIN maxes ON (apex.service_class = maxes.service_class AND apex.service_class_slots =
    maxes.max_service_class_slots)
    GROUP BY apex.service_class, apex.num_query_tasks, maxes.d, maxes.dt_h
    ORDER BY apex.service_class, maxes.d, maxes.dt_h;

```

The `max_service_class_slots` column represents the maximum number of WLM query slots in the query queue for that hour. If underutilized queues exist, implement the slot reduction optimization by [modifying a parameter group](#), as described in the *Amazon Redshift Management Guide*.

Implementation tips

- If your workload is highly variable in volume, make sure that the analysis captured a peak utilization period. If it didn't, run the preceding SQL repeatedly to monitor peak concurrency requirements.
- For more details on interpreting the query results from the preceding SQL code, see the [wlm_apex_hourly.sql script](#) on GitHub.

Skip compression analysis during COPY

When you load data into an empty table with compression encoding declared with the `COPY` command, Amazon Redshift applies storage compression. This optimization ensures that data in your cluster is stored efficiently even when loaded by end users. The analysis required to apply compression can require significant time.

Analysis

The Advisor analysis checks for `COPY` operations that were delayed by automatic compression analysis. The analysis determines the compression encodings by sampling the data while it's being loaded. This sampling is similar to that performed by the [ANALYZE COMPRESSION \(p. 665\)](#) command.

When you load data as part of a structured process, such as in an overnight extract, transform, load (ETL) batch, you can define the compression beforehand. You can also optimize your table definitions to skip this phase permanently without any negative impacts.

Recommendation

To improve COPY responsiveness by skipping the compression analysis phase, implement either of the following two options:

- Use the column ENCODE parameter when creating any tables that you load using the COPY command.
- Turn off compression altogether by supplying the COMPUPDATE OFF parameter in the COPY command.

The best solution is generally to use column encoding during table creation, because this approach also maintains the benefit of storing compressed data on disk. You can use the ANALYZE COMPRESSION command to suggest compression encodings, but you must recreate the table to apply these encodings. To automate this process, you can use the AWS[ColumnEncodingUtility](#), found on GitHub.

To identify recent COPY operations that triggered automatic compression analysis, run the following SQL command.

```
WITH xids AS (
    SELECT xid FROM stl_query WHERE userid>1 AND aborted=0
    AND querytxt = 'analyze compression phase 1' GROUP BY xid
    INTERSECT SELECT xid FROM stl_commit_stats WHERE node=-1)
SELECT a.userid, a.query, a.xid, astarttime, b.complyze_sec,
    a.copy_sec, a.copy_sql
FROM (SELECT q.userid, q.query, q.xid, date_trunc('s',q.starttime)
    starttime, substring(querytxt,1,100) as copy_sql,
    ROUND(datediff(ms,starttime,endtime)::numeric / 1000.0, 2) copy_sec
    FROM stl_query q JOIN xids USING (xid)
    WHERE (querytxt ilike 'copy %from%' OR querytxt ilike '% copy %from%')
        AND querytxt not like 'COPY ANALYZE %') a
LEFT JOIN (SELECT xid,
    ROUND(sum(datediff(ms,starttime,endtime))::numeric / 1000.0,2) complyze_sec
    FROM stl_query q JOIN xids USING (xid)
    WHERE (querytxt like 'COPY ANALYZE %'
        OR querytxt like 'analyze compression phase %')
        GROUP BY xid ) b ON a.xid = b.xid
WHERE b.complyze_sec IS NOT NULL ORDER BY a.copy_sql, a.starttime;
```

Implementation tips

- Ensure that all tables of significant size created during your ETL processes (for example, staging tables and temporary tables) declare a compression encoding for all columns except the first sort key.
- Estimate the expected lifetime size of the table being loaded for each of the COPY commands identified by the SQL command preceding. If you are confident that the table will remain extremely small, turn off compression altogether with the COMPUPDATE OFF parameter. Otherwise, create the table with explicit compression before loading it with the COPY command.

Split Amazon S3 objects loaded by COPY

The COPY command takes advantage of the massively parallel processing (MPP) architecture in Amazon Redshift to read and load data from files on Amazon S3. The COPY command loads the data in parallel from multiple files, dividing the workload among the nodes in your cluster. To achieve optimal throughput, we strongly recommend that you divide your data into multiple files to take advantage of parallel processing.

Analysis

The Advisor analysis identifies COPY commands that load large datasets contained in a small number of files staged in Amazon S3. Long-running COPY commands that load large datasets from a few files often have an opportunity for considerable performance improvement. When Advisor identifies that these COPY commands are taking a significant amount of time, it creates a recommendation to increase parallelism by splitting the data into additional files in Amazon S3.

Recommendation

In this case, we recommend the following actions, listed in priority order:

1. Optimize COPY commands that load fewer files than the number of cluster nodes.
2. Optimize COPY commands that load fewer files than the number of cluster slices.
3. Optimize COPY commands where the number of files is not a multiple of the number of cluster slices.

Certain COPY commands load a significant amount of data or run for a significant duration. For these commands, we recommend that you load a number of data objects from Amazon S3 that is equivalent to a multiple of the number of slices in the cluster. To identify how many S3 objects each COPY command has loaded, run the following SQL code as a superuser.

```
SELECT
    query, COUNT(*) num_files,
    ROUND(MAX(wq.total_exec_time/1000000.0),2) execution_secs,
    ROUND(SUM(transfer_size)/(1024.0*1024.0),2) total_mb,
    SUBSTRING(querytxt,1,60) copy_sql
FROM stl_s3client s
JOIN stl_query q USING (query)
JOIN stl_wlm_query wq USING (query)
WHERE s.userid>1 AND http_method = 'GET'
    AND POSITION('COPY ANALYZE' IN querytxt) = 0
    AND aborted = 0 AND final_state='Completed'
GROUP BY query, querytxt
HAVING (SUM(transfer_size)/(1024*1024))/COUNT(*) >= 2
ORDER BY CASE
    WHEN COUNT(*) < (SELECT max(node)+1 FROM stv_slices) THEN 1
    WHEN COUNT(*) < (SELECT COUNT(*) FROM stv_slices WHERE node=0) THEN 2
    ELSE 2+((COUNT(*) % (SELECT COUNT(*) FROM stv_slices))/(SELECT COUNT(*)::DECIMAL FROM
    stv_slices))
END, (SUM(transfer_size)/(1024.0*1024.0))/COUNT(*) DESC;
```

Implementation tips

- The number of slices in a node depends on the node size of the cluster. For more information about the number of slices in the various node types, see [Clusters and Nodes in Amazon Redshift](#) in the *Amazon Redshift Management Guide*.
- You can load multiple files by specifying a common prefix, or prefix key, for the set, or by explicitly listing the files in a manifest file. For more information about loading files, see [Loading data from compressed and uncompressed files \(p. 66\)](#).
- Amazon Redshift doesn't take file size into account when dividing the workload. Split your load data files so that the files are about equal size, between 1 MB and 1 GB after compression.

Update table statistics

Amazon Redshift uses a cost-based query optimizer to choose the optimum execution plan for queries. The cost estimates are based on table statistics gathered using the ANALYZE command. When statistics

are out of date or missing, the database might choose a less efficient plan for query execution, especially for complex queries. Maintaining current statistics helps complex queries run in the shortest possible time.

Analysis

The Advisor analysis tracks tables whose statistics are out-of-date or missing. It reviews table access metadata associated with complex queries. If tables that are frequently accessed with complex patterns are missing statistics, Advisor creates a **critical** recommendation to run ANALYZE. If tables that are frequently accessed with complex patterns have out-of-date statistics, Advisor creates a **suggested** recommendation to run ANALYZE.

Recommendation

Whenever table content changes significantly, update statistics with ANALYZE. We recommend running ANALYZE whenever a significant number of new data rows are loaded into an existing table with COPY or INSERT commands. We also recommend running ANALYZE whenever a significant number of rows are modified using UPDATE or DELETE commands. To identify tables with missing or out-of-date statistics, run the following SQL command as a superuser. The results are ordered from largest to smallest table.

To identify tables with missing or out-of-date statistics, run the following SQL command as a superuser. The results are ordered from largest to smallest table.

```
SELECT
    ti.schema||'.'||ti."table" tablename,
    ti.size table_size_mb,
    ti.stats_off statistics_accuracy
FROM svv_table_info ti
WHERE ti.stats_off > 5.00
ORDER BY ti.size DESC;
```

Implementation tips

The default ANALYZE threshold is 10 percent. This default means that the ANALYZE command skips a given table if fewer than 10 percent of the table's rows have changed since the last ANALYZE. As a result, you might choose to issue ANALYZE commands at the end of each ETL process. Taking this approach means that ANALYZE is often skipped but also ensures that ANALYZE runs when needed.

ANALYZE statistics have the most impact for columns that are used in joins (for example, JOIN `tbl_a` ON `col_b`) or as predicates (for example, WHERE `col_b = 'xyz'`). By default, ANALYZE collects statistics for all columns in the table specified. If needed, you can reduce the time required to run ANALYZE by running ANALYZE only for the columns where it has the most impact. You can run the following SQL command to identify columns used as predicates. You can also let Amazon Redshift choose which columns to analyze by specifying ANALYZE PREDICATE COLUMNS.

```
WITH predicate_column_info AS (
SELECT ns.nspname AS schema_name, c.relname AS table_name, a.attnum AS col_num, a.attname
    AS col_name,
    CASE
        WHEN 10002 = s.stakind1 THEN array_to_string(stavalue1, '||')
        WHEN 10002 = s.stakind2 THEN array_to_string(stavalue2, '||')
        WHEN 10002 = s.stakind3 THEN array_to_string(stavalue3, '||')
        WHEN 10002 = s.stakind4 THEN array_to_string(stavalue4, '||')
        ELSE NULL::varchar
    END AS pred_ts
FROM pg_statistic s
JOIN pg_class c ON c.oid = s.starelid
JOIN pg_namespace ns ON c.relnamespace = ns.oid
```

```
JOIN pg_attribute a ON c.oid = a.attrelid AND a.attnum = s.staattnum)
SELECT schema_name, table_name, col_num, col_name,
       pred_ts NOT LIKE '2000-01-01%' AS is_predicate,
       CASE WHEN pred_ts NOT LIKE '2000-01-01%' THEN (split_part(pred_ts,
'|||',1))::timestamp ELSE NULL::timestamp END as first_predicate_use,
       CASE WHEN pred_ts NOT LIKE '%||2000-01-01%' THEN (split_part(pred_ts,
'|||',2))::timestamp ELSE NULL::timestamp END as last_analyze
FROM predicate_column_info;
```

For more information, see [Analyzing tables \(p. 105\)](#).

Enable short query acceleration

Short query acceleration (SQA) prioritizes selected short-running queries ahead of longer-running queries. SQA runs short-running queries in a dedicated space, so that SQA queries aren't forced to wait in queues behind longer queries. SQA only prioritizes queries that are short-running and are in a user-defined queue. With SQA, short-running queries begin running more quickly and users see results sooner.

If you turn on SQA, you can reduce or eliminate workload management (WLM) queues that are dedicated to running short queries. In addition, long-running queries don't need to contend with short queries for slots in a queue, so you can configure your WLM queues to use fewer query slots. When you use lower concurrency, query throughput is increased and overall system performance is improved for most workloads. For more information, see [Working with short query acceleration \(p. 518\)](#).

Analysis

Advisor checks for workload patterns and reports the number of recent queries where SQA would reduce latency and the daily queue time for SQA-eligible queries.

Recommendation

Modify the WLM configuration to turn on SQA. Amazon Redshift uses a machine learning algorithm to analyze each eligible query. Predictions improve as SQA learns from your query patterns. For more information, see [Configuring Workload Management](#).

When you turn on SQA, WLM sets the maximum runtime for short queries to dynamic by default. We recommend keeping the dynamic setting for SQA maximum runtime.

Implementation tips

To check whether SQA is turned on, run the following query. If the query returns a row, then SQA is turned on.

```
select * from stv_wlm_service_class_config
where service_class = 14;
```

For more information, see [Monitoring SQA \(p. 519\)](#).

Alter distribution keys on tables

Amazon Redshift distributes table rows throughout the cluster according to the table distribution style. Tables with KEY distribution require a column as the distribution key (DISTKEY). A table row is assigned to a node slice of a cluster based on its DISTKEY column value.

An appropriate DISTKEY places a similar number of rows on each node slice and is frequently referenced in join conditions. An optimized join occurs when tables are joined on their DISTKEY columns, accelerating query performance.

Analysis

Advisor analyzes your cluster's workload to identify the most appropriate distribution key for the tables that can significantly benefit from a KEY distribution style.

Recommendation

Advisor provides [ALTER TABLE \(p. 640\)](#) statements that alter the DISTSTYLE and DISTKEY of a table based on its analysis. To realize a significant performance benefit, make sure to implement all SQL statements within a recommendation group.

Redistributing a large table with ALTER TABLE consumes cluster resources and requires temporary table locks at various times. Implement each recommendation group when other cluster workload is light. For more details on optimizing table distribution properties, see the [Amazon Redshift Engineering's Advanced Table Design Playbook: Distribution Styles and Distribution Keys](#).

For more information about ALTER DISTSTYLE and DISTKEY, see [ALTER TABLE \(p. 640\)](#).

Note

If you don't see a recommendation that doesn't necessarily mean that the current distribution styles are the most appropriate. Advisor doesn't provide recommendations when there isn't enough data or the expected benefit of redistribution is small.

Advisor recommendations apply to a particular table and don't necessarily apply to a table that contains a column with the same name. Tables that share a column name can have different characteristics for those columns unless data inside the tables is the same.

If you see recommendations for staging tables that are created or dropped by ETL jobs, modify your ETL processes to use the Advisor recommended distribution keys.

Alter sort keys on tables

Amazon Redshift sorts table rows according to the table [sort key \(p. 60\)](#). The sorting of table rows is based on the sort key column values.

Sorting a table on an appropriate sort key can accelerate performance of queries, especially those with range-restricted predicates, by requiring fewer table blocks to be read from disk.

Analysis

Advisor analyzes your cluster's workload over several days to identify a beneficial sort key for your tables.

Recommendation

Advisor provides two groups of ALTER TABLE statements that alter the sort key of a table based on its analysis:

- Statements that alter a table that currently doesn't have a sort key to add a COMPOUND sort key.
- Statements that alter a sort key from INTERLEAVED to COMPOUND or no sort key.

Using compound sort keys significantly reduces maintenance overhead. Tables with compound sort keys don't need the expensive VACUUM REINDEX operations that are necessary for interleaved sorts. In practice, compound sort keys are more effective than interleaved sort keys for the vast majority of Amazon Redshift workloads. However, if a table is small, it's more efficient not to have a sort key to avoid sort key storage overhead.

When sorting a large table with the ALTER TABLE, cluster resources are consumed and table locks are required at various times. Implement each recommendation when a cluster's workload is moderate. More details on optimizing table sort key configurations can be found in the [Amazon Redshift Engineering's Advanced Table Design Playbook: Compound and Interleaved Sort Keys](#).

For more information about ALTER SORTKEY, see [ALTER TABLE \(p. 640\)](#).

Note

If you don't see a recommendation for a table, that doesn't necessarily mean that the current configuration is the best. Advisor doesn't provide recommendations when there isn't enough data or the expected benefit of sorting is small.

Advisor recommendations apply to a particular table and don't necessarily apply to a table that contains a column with the same name and data type. Tables that share column names can have different recommendations based on the data in the tables and the workload.

Alter compression encodings on columns

Compression is a column-level operation that reduces the size of data when it's stored. Compression is used in Amazon Redshift to conserve storage space and improve query performance by reducing the amount of disk I/O. We recommend an optimal compression encoding for each column based on its data type and on query patterns. With optimal compression, queries can run more efficiently and the database can take up minimal storage space.

Analysis

Advisor performs analysis of your cluster's workload and database schema continually to identify the optimal compression encoding for each table column.

Recommendation

Advisor provides ALTER TABLE statements that change the compression encoding of particular columns, based on its analysis.

Changing column compression encodings with [ALTER TABLE \(p. 640\)](#) consumes cluster resources and requires table locks at various times. It's best to implement recommendations when the cluster workload is light.

For reference, [ALTER TABLE examples \(p. 649\)](#) shows several statements that change the encoding for a column.

Note

Advisor doesn't provide recommendations when there isn't enough data or the expected benefit of changing the encoding is small.

Data type recommendations

Amazon Redshift has a library of SQL data types for various use cases. These include integer types like INT and types to store characters, like VARCHAR. Redshift stores types in an optimized way to provide fast access and good query performance. Also, Redshift provides functions for specific types, which you can use to format or perform calculations on query results.

Analysis

Advisor performs analysis of your cluster's workload and database schema continually to identify columns that can benefit significantly from a data type change.

Recommendation

Advisor provides an ALTER TABLE statement that adds a new column with the suggested data type. An accompanying UPDATE statement copies data from the existing column to the new column. After you create the new column and load the data, change your queries and ingestion scripts to access the new column. Then leverage features and functions specialized to the new data type, found in [SQL functions reference \(p. 1020\)](#).

Copying existing data to the new column can take time. We recommend that you implement each advisor recommendation when the cluster's workload is light. Reference the list of available data types at [Data types \(p. 574\)](#).

Note that Advisor doesn't provide recommendations when there isn't enough data or the expected benefit of changing the data type is small.

Tutorials for Amazon Redshift

Follow the steps in these tutorials to learn about Amazon Redshift features:

- [Tutorial: Loading data from Amazon S3 \(p. 124\)](#)
- [Tutorial: Querying nested data with Amazon Redshift Spectrum \(p. 294\)](#)
- [Tutorial: Configuring manual workload management \(WLM\) queues \(p. 505\)](#)
- [Tutorial: Using spatial SQL functions with Amazon Redshift \(p. 222\)](#)
- [Tutorials for Amazon Redshift ML \(p. 406\)](#)

Working with automatic table optimization

Automatic table optimization is a self-tuning capability that automatically optimizes the design of tables by applying sort and distribution keys without the need for administrator intervention. By using automation to tune the design of tables, you can get started and get the fastest performance without investing time to manually tune and implement table optimizations.

Automatic table optimization continuously observes how queries interact with tables. It uses advanced artificial intelligence methods to choose sort and distribution keys to optimize performance for the cluster's workload. If Amazon Redshift determines that applying a key improves cluster performance, tables are automatically altered within hours from the time the cluster was created, with minimal impact to queries.

To take advantage of this automation, an Amazon Redshift administrator creates a new table, or alters an existing table to enable it to use automatic optimization. Existing tables with a distribution style or sort key of AUTO are already enabled for automation. When you run queries against those tables, Amazon Redshift determines if a sort key or distribution key will improve performance. If so, then Amazon Redshift automatically modifies the table without requiring administrator intervention. If a minimum number of queries are run, optimizations are applied within hours of the cluster being launched.

If Amazon Redshift determines that a distribution key improves the performance of queries, tables where distribution style is AUTO can have their distribution style changed to KEY.

Topics

- [Enabling automatic table optimization \(p. 36\)](#)
- [Removing automatic table optimization from a table \(p. 37\)](#)
- [Monitoring actions of automatic table optimization \(p. 37\)](#)
- [Working with column compression \(p. 38\)](#)
- [Working with data distribution styles \(p. 49\)](#)
- [Working with sort keys \(p. 60\)](#)
- [Defining table constraints \(p. 62\)](#)

Enabling automatic table optimization

By default, tables created without explicitly defining sort keys or distributions keys are set to AUTO. At the time of table creation, you can also explicitly set a sort or a distribution key manually. If you set the sort or distribution key, then the table is not automatically managed.

To enable an existing table to be automatically optimized, use the ALTER statement options to change the table to AUTO. You might choose to define automation for sort keys, but not for distribution keys (and vice versa). If you run an ALTER statement to convert a table to be an automated table, existing sort keys and distribution styles are preserved.

```
ALTER TABLE table_name ALTER SORTKEY AUTO;
```

```
ALTER TABLE table_name ALTER DISTSTYLE AUTO;
```

For more information, see [ALTER TABLE \(p. 640\)](#).

Initially, a table has no distribution key or sort key. The distribution style is set to either EVEN or ALL depending on table size. As the table grows in size, Amazon Redshift applies the optimal distribution keys and sort keys. Optimizations are applied within hours after a minimum number of queries are run. When determining sort key optimizations, Amazon Redshift attempts to optimize the data blocks read from disk during a table scan. When determining distribution style optimizations, Amazon Redshift tries to optimize the number of bytes transferred between cluster nodes.

Removing automatic table optimization from a table

You can remove a table from automatic optimization. Removing a table from automation involves selecting a sort key or distribution style. To change distribution style, specify a specific distribution style.

```
ALTER TABLE table_name ALTER DISTSTYLE EVEN;
```

```
ALTER TABLE table_name ALTER DISTSTYLE ALL;
```

```
ALTER TABLE table_name ALTER DISTSTYLE KEY DISTKEY c1;
```

To change a sort key, you can define a sort key or choose none.

```
ALTER TABLE table_name ALTER SORTKEY(c1, c2);
```

```
ALTER TABLE table_name ALTER SORTKEY NONE;
```

Monitoring actions of automatic table optimization

The system view SVV_ALTER_TABLE_RECOMMENDATIONS records the current Amazon Redshift Advisor recommendations for tables. This view shows recommendations for all tables, those that are defined for automatic optimization and those that aren't.

To view if a table is defined for automatic optimization, query the system view SVV_TABLE_INFO. Entries appear only for tables visible in the current session's database. Recommendations are inserted into the view twice per day starting within hours from the time the cluster was created. After a recommendation is available, it's started within an hour. After a recommendation has been applied (either by Amazon Redshift or by you), it no longer appears in the view.

The system view SVL_AUTO_WORKER_ACTION shows an audit log of all the actions taken by the Amazon Redshift, and the previous state of the table.

The system view SVV_TABLE_INFO lists all of the tables in the system, along with a column to indicate whether the sort key and distribution style of the table is set to AUTO.

For more information about these system views, see [System monitoring \(provisioned only\) \(p. 1494\)](#).

Working with column compression

Compression is a column-level operation that reduces the size of data when it is stored. Compression conserves storage space and reduces the size of data that is read from storage, which reduces the amount of disk I/O and therefore improves query performance.

ENCODE AUTO is the default for tables. When a table is set to ENCODE AUTO, Amazon Redshift automatically manages compression encoding for all columns in the table. For more information, see [CREATE TABLE \(p. 823\)](#) and [ALTER TABLE \(p. 640\)](#).

However, if you specify compression encoding for any column in the table, the table is no longer set to ENCODE AUTO. Amazon Redshift no longer automatically manages compression encoding for all columns in the table.

You can apply a compression type, or *encoding*, to the columns in a table manually when you create the table. Or you can use the COPY command to analyze and apply compression automatically. For more information, see [Let COPY choose compression encodings \(p. 17\)](#). For details about applying automatic compression, see [Loading tables with automatic compression \(p. 87\)](#).

Note

We strongly recommend using the COPY command to apply automatic compression.

You might choose to apply compression encodings manually if the new table shares the same data characteristics as another table. Or you might do so if you discover in testing that the compression encodings applied during automatic compression are not the best fit for your data. If you choose to apply compression encodings manually, you can run the [ANALYZE COMPRESSION \(p. 665\)](#) command against an already populated table and use the results to choose compression encodings.

To apply compression manually, you specify compression encodings for individual columns as part of the CREATE TABLE statement. The syntax is as follows.

```
CREATE TABLE table_name (column_name  
data_type ENCODE encoding-type)[, ...]
```

Here, *encoding-type* is taken from the keyword table in the following section.

For example, the following statement creates a two-column table, PRODUCT. When data is loaded into the table, the PRODUCT_ID column is not compressed, but the PRODUCT_NAME column is compressed, using the byte dictionary encoding (BYTEDICT).

```
create table product(  
product_id int encode raw,  
product_name char(20) encode bytedict);
```

You can specify the encoding for a column when it is added to a table using the ALTER TABLE command.

```
ALTER TABLE table-name ADD [ COLUMN ] column_name column_type ENCODE encoding-type
```

Topics

- [Compression encodings \(p. 39\)](#)
- [Testing compression encodings \(p. 45\)](#)
- [Example: Choosing compression encodings for the CUSTOMER table \(p. 47\)](#)

Compression encodings

A *compression encoding* specifies the type of compression that is applied to a column of data values as rows are added to a table.

ENCODE AUTO is the default for tables. When a table is set to ENCODE AUTO, Amazon Redshift automatically manages compression encoding for all columns in the table. For more information, see [CREATE TABLE \(p. 823\)](#) and [ALTER TABLE \(p. 640\)](#).

However, if you specify compression encoding for any column in the table, the table is no longer set to ENCODE AUTO. Amazon Redshift no longer automatically manages compression encoding for all columns in the table.

When you use CREATE TABLE, ENCODE AUTO is disabled when you specify compression encoding for any column in the table. If ENCODE AUTO is disabled, Amazon Redshift automatically assigns compression encoding to columns for which you don't specify an ENCODE type as follows:

- Columns that are defined as sort keys are assigned RAW compression.
- Columns that are defined as BOOLEAN, REAL, or DOUBLE PRECISION data types are assigned RAW compression.
- Columns that are defined as SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIMESTAMP, or Timestamptz data types are assigned AZ64 compression.
- Columns that are defined as CHAR or VARCHAR data types are assigned LZO compression.

You can change a table's encoding after creating it by using ALTER TABLE. If you disable ENCODE AUTO using ALTER TABLE, Amazon Redshift no longer automatically manages compression encodings for your columns. All columns will keep the compression encoding types that they had when you disabled ENCODE AUTO until you change them or you enable ENCODE AUTO again.

The following table identifies the supported compression encodings and the data types that support the encoding.

Encoding type	Keyword in CREATE TABLE and ALTER TABLE	Data types
Raw (no compression)	RAW	All
AZ64	AZ64	SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIMESTAMP, Timestamptz
Byte dictionary	BYTEDICT	SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, CHAR, VARCHAR, DATE, TIMESTAMP, Timestamptz
Delta	DELTA DELTA32K	SMALLINT, INT, BIGINT, DATE, TIMESTAMP, DECIMAL INT, BIGINT, DATE, TIMESTAMP, DECIMAL
LZO	LZO	SMALLINT, INTEGER, BIGINT, DECIMAL, CHAR, VARCHAR, DATE, TIMESTAMP, Timestamptz, SUPER

Encoding type	Keyword in CREATE TABLE and ALTER TABLE	Data types
Mostlyn	MOSTLY8	SMALLINT, INT, BIGINT, DECIMAL
	MOSTLY16	INT, BIGINT, DECIMAL
	MOSTLY32	BIGINT, DECIMAL
Run-length	RUNLENGTH	SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP, Timestamptz
Text	TEXT255	VARCHAR only
	TEXT32K	VARCHAR only
Zstandard	ZSTD	SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP, Timestamptz, SUPER

Raw encoding

Raw encoding is the default encoding for columns that are designated as sort keys and columns that are defined as BOOLEAN, REAL, or DOUBLE PRECISION data types. With raw encoding, data is stored in raw, uncompressed form.

AZ64 encoding

AZ64 is a proprietary compression encoding algorithm designed by Amazon to achieve a high compression ratio and improved query processing. At its core, the AZ64 algorithm compresses smaller groups of data values and uses single instruction, multiple data (SIMD) instructions for parallel processing. Use AZ64 to achieve significant storage savings and high performance for numeric, date, and time data types.

You can use AZ64 as the compression encoding when defining columns using CREATE TABLE and ALTER TABLE statements with the following data types:

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- DATE
- TIMESTAMP
- Timestamptz

Byte-dictionary encoding

In byte dictionary encoding, a separate dictionary of unique values is created for each block of column values on disk. (An Amazon Redshift disk block occupies 1 MB.) The dictionary contains up to 256 one-byte values that are stored as indexes to the original data values. If more than 256 values are stored in a single block, the extra values are written into the block in raw, uncompressed form. The process repeats for each disk block.

This encoding is very effective on low cardinality string columns. This encoding is optimal when the data domain of a column is fewer than 256 unique values.

For columns with the string data type (CHAR and VARCHAR) encoded with BYTEDICT, Amazon Redshift performs vectorized scans and predicate evaluations that operate over compressed data directly. These scans use hardware-specific single instruction and multiple data (SIMD) instructions for parallel processing. This significantly speeds up the scanning of string columns. Byte-dictionary encoding is especially space-efficient if a CHAR/VARCHAR column holds long character strings.

Suppose that a table has a COUNTRY column with a CHAR(30) data type. As data is loaded, Amazon Redshift creates the dictionary and populates the COUNTRY column with the index value. The dictionary contains the indexed unique values, and the table itself contains only the one-byte subscripts of the corresponding values.

Note

Trailing blanks are stored for fixed-length character columns. Therefore, in a CHAR(30) column, every compressed value saves 29 bytes of storage when you use the byte-dictionary encoding.

The following table represents the dictionary for the COUNTRY column.

Unique data value	Dictionary index	Size (fixed length, 30 bytes per value)
England	0	30
United States of America	1	30
Venezuela	2	30
Sri Lanka	3	30
Argentina	4	30
Japan	5	30
Total		180

The following table represents the values in the COUNTRY column.

Original data value	Original size (fixed length, 30 bytes per value)	Compressed value (index)	New size (bytes)
England	30	0	1
England	30	0	1
United States of America	30	1	1
United States of America	30	1	1
Venezuela	30	2	1
Sri Lanka	30	3	1
Argentina	30	4	1

Original data value	Original size (fixed length, 30 bytes per value)	Compressed value (index)	New size (bytes)
Japan	30	5	1
Sri Lanka	30	3	1
Argentina	30	4	1
Total	300		10

The total compressed size in this example is calculated as follows: 6 different entries are stored in the dictionary ($6 * 30 = 180$), and the table contains 10 1-byte compressed values, for a total of 190 bytes.

Delta encoding

Delta encodings are very useful for date time columns.

Delta encoding compresses data by recording the difference between values that follow each other in the column. This difference is recorded in a separate dictionary for each block of column values on disk. (An Amazon Redshift disk block occupies 1 MB.) For example, suppose that the column contains 10 integers in sequence from 1 to 10. The first are stored as a 4-byte integer (plus a 1-byte flag). The next nine are each stored as a byte with the value 1, indicating that it is one greater than the previous value.

Delta encoding comes in two variations:

- DELTA records the differences as 1-byte values (8-bit integers)
- DELTA32K records differences as 2-byte values (16-bit integers)

If most of the values in the column could be compressed by using a single byte, the 1-byte variation is very effective. However, if the deltas are larger, this encoding, in the worst case, is somewhat less effective than storing the uncompressed data. Similar logic applies to the 16-bit version.

If the difference between two values exceeds the 1-byte range (DELTA) or 2-byte range (DELTA32K), the full original value is stored, with a leading 1-byte flag. The 1-byte range is from -127 to 127, and the 2-byte range is from -32K to 32K.

The following table shows how a delta encoding works for a numeric column.

Original data value	Original size (bytes)	Difference (delta)	Compressed value	Compressed size (bytes)
1	4		1	1+4 (flag + actual value)
5	4	4	4	1
50	4	45	45	1
200	4	150	150	1+4 (flag + actual value)
185	4	-15	-15	1
220	4	35	35	1
221	4	1	1	1

Original data value	Original size (bytes)	Difference (delta)	Compressed value	Compressed size (bytes)
Totals	28			15

LZO encoding

LZO encoding provides a very high compression ratio with good performance. LZO encoding works especially well for CHAR and VARCHAR columns that store very long character strings. They are especially good for free-form text, such as product descriptions, user comments, or JSON strings.

Mostly encoding

Mostly encodings are useful when the data type for a column is larger than most of the stored values require. By specifying a mostly encoding for this type of column, you can compress the majority of the values in the column to a smaller standard storage size. The remaining values that cannot be compressed are stored in their raw form. For example, you can compress a 16-bit column, such as an INT2 column, to 8-bit storage.

In general, the mostly encodings work with the following data types:

- SMALLINT/INT2 (16-bit)
- INTEGER/INT (32-bit)
- BIGINT/INT8 (64-bit)
- DECIMAL/NUMERIC (64-bit)

Choose the appropriate variation of the mostly encoding to suit the size of the data type for the column. For example, apply MOSTLY8 to a column that is defined as a 16-bit integer column. Applying MOSTLY16 to a column with a 16-bit data type or MOSTLY32 to a column with a 32-bit data type is disallowed.

Mostly encodings might be less effective than no compression when a relatively high number of the values in the column can't be compressed. Before applying one of these encodings to a column, perform a check. *Most* of the values that you are going to load now (and are likely to load in the future) should fit into the ranges shown in the following table.

Encoding	Compressed storage size	Range of values that can be compressed (values outside the range are stored raw)
MOSTLY8	1 byte (8 bits)	-128 to 127
MOSTLY16	2 bytes (16 bits)	-32768 to 32767
MOSTLY32	4 bytes (32 bits)	-2147483648 to +2147483647

Note

For decimal values, ignore the decimal point to determine whether the value fits into the range. For example, 1,234.56 is treated as 123,456 and can be compressed in a MOSTLY32 column.

For example, the VENUEID column in the VENUE table is defined as a raw integer column, which means that its values consume 4 bytes of storage. However, the current range of values in the column is **0 to 309**. Therefore, recreating and reloading this table with MOSTLY16 encoding for VENUEID would reduce the storage of every value in that column to 2 bytes.

If the VENUEID values referenced in another table were mostly in the range of 0 to 127, it might make sense to encode that foreign-key column as MOSTLY8. Before making the choice, run several queries against the referencing table data to find out whether the values mostly fall into the 8-bit, 16-bit, or 32-bit range.

The following table shows compressed sizes for specific numeric values when the MOSTLY8, MOSTLY16, and MOSTLY32 encodings are used:

Original value	Original INT or BIGINT size (bytes)	MOSTLY8 compressed size (bytes)	MOSTLY16 compressed size (bytes)	MOSTLY32 compressed size (bytes)	
1	4	1	2	4	
10	4	1	2	4	
100	4	1	2	4	
1000	4	Same as raw data size	2	4	
10000	4		2	4	
20000	4		2	4	
40000	8		Same as raw data size	4	
100000	8			4	
2000000000	8			4	

Run length encoding

Run length encoding replaces a value that is repeated consecutively with a token that consists of the value and a count of the number of consecutive occurrences (the length of the run). A separate dictionary of unique values is created for each block of column values on disk. (An Amazon Redshift disk block occupies 1 MB.) This encoding is best suited to a table in which data values are often repeated consecutively, for example, when the table is sorted by those values.

For example, suppose that a column in a large dimension table has a predictably small domain, such as a COLOR column with fewer than 10 possible values. These values are likely to fall in long sequences throughout the table, even if the data is not sorted.

We don't recommend applying run length encoding on any column that is designated as a sort key. Range-restricted scans perform better when blocks contain similar numbers of rows. If sort key columns are compressed much more highly than other columns in the same query, range-restricted scans might perform poorly.

The following table uses the COLOR column example to show how the run length encoding works.

Original data value	Original size (bytes)	Compressed value (token)	Compressed size (bytes)
Blue	4	{2,Blue}	5
Blue	4		0
Green	5	{3,Green}	6

Original data value	Original size (bytes)	Compressed value (token)	Compressed size (bytes)
Green	5		0
Green	5		0
Blue	4	{1,Blue}	5
Yellow	6		7
Yellow	6		0
Yellow	6		0
Yellow	6		0
Total	51		23

Text255 and Text32k encodings

Text255 and text32k encodings are useful for compressing VARCHAR columns in which the same words recur often. A separate dictionary of unique words is created for each block of column values on disk. (An Amazon Redshift disk block occupies 1 MB.) The dictionary contains the first 245 unique words in the column. Those words are replaced on disk by a one-byte index value representing one of the 245 values, and any words that are not represented in the dictionary are stored uncompressed. The process repeats for each 1-MB disk block. If the indexed words occur frequently in the column, the column yields a high compression ratio.

For the text32k encoding, the principle is the same, but the dictionary for each block does not capture a specific number of words. Instead, the dictionary indexes each unique word it finds until the combined entries reach a length of 32K, minus some overhead. The index values are stored in two bytes.

For example, consider the VENUENAME column in the VENUE table. Words such as **Arena**, **Center**, and **Theatre** recur in this column and are likely to be among the first 245 words encountered in each block if text255 compression is applied. If so, this column benefits from compression. This is because every time those words appear, they occupy only 1 byte of storage (instead of 5, 6, or 7 bytes, respectively).

Zstandard encoding

Zstandard (ZSTD) encoding provides a high compression ratio with very good performance across diverse datasets. ZSTD works especially well with CHAR and VARCHAR columns that store a wide range of long and short strings, such as product descriptions, user comments, logs, and JSON strings. Where some algorithms, such as [Delta \(p. 42\)](#) encoding or [Mostly \(p. 43\)](#) encoding, can potentially use more storage space than no compression, ZSTD is very unlikely to increase disk usage.

ZSTD supports SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP, and TIMESTAMPTZ data types.

Testing compression encodings

If you decide to manually specify column encodings, you might want to test different encodings with your data.

Note

We recommend that you use the COPY command to load data whenever possible, and allow the COPY command to choose the optimal encodings based on your data. Or you can use the

[ANALYZE COMPRESSION \(p. 665\)](#) command to view the suggested encodings for existing data. For details about applying automatic compression, see [Loading tables with automatic compression \(p. 87\)](#).

To perform a meaningful test of data compression, you must have a large number of rows. For this example, we create a table and insert rows by using a statement that selects from two tables; VENUE and LISTING. We leave out the WHERE clause that would normally join the two tables. The result is that *each* row in the VENUE table is joined to *all* of the rows in the LISTING table, for a total of over 32 million rows. This is known as a Cartesian join and normally is not recommended. However, for this purpose, it's a convenient method of creating many rows. If you have an existing table with data that you want to test, you can skip this step.

After we have a table with sample data, we create a table with seven columns. Each has a different compression encoding: raw, bytedict, lzo, run length, text255, text32k, and zstd. We populate each column with exactly the same data by running an INSERT command that selects the data from the first table.

To test compression encodings, do the following:

1. (Optional) First, use a Cartesian join to create a table with a large number of rows. Skip this step if you want to test an existing table.

```
create table cartesian_venue(
    venueid smallint not null distkey sortkey,
    venuename varchar(100),
    venuecity varchar(30),
    venuestate char(2),
    venueseats integer);

insert into cartesian_venue
select venueid, venuename, venuecity, venuestate, venueseats
from venue, listing;
```

2. Next, create a table with the encodings that you want to compare.

```
create table encodingvenue (
    venueraw varchar(100) encode raw,
    venuebytedict varchar(100) encode bytedict,
    venuelzo varchar(100) encode lzo,
    venuerunlength varchar(100) encode runlength,
    venuetext255 varchar(100) encode text255,
    venuetext32k varchar(100) encode text32k,
    venuezstd varchar(100) encode zstd);
```

3. Insert the same data into all of the columns using an INSERT statement with a SELECT clause.

```
insert into encodingvenue
select venuename as venueraw, venuename as venuebytedict, venuename as venuelzo,
    venuename as venuerunlength, venuename as venuetext32k, venuename as venuetext255,
    venuename as venuezstd
from cartesian_venue;
```

4. Verify the number of rows in the new table.

```
select count(*) from encodingvenue

count
-----
38884394
(1 row)
```

5. Query the [STV_BLOCKLIST \(p. 1579\)](#) system table to compare the number of 1 MB disk blocks used by each column.

The MAX aggregate function returns the highest block number for each column. The STV_BLOCKLIST table includes details for three system-generated columns. This example uses `col < 6` in the WHERE clause to exclude the system-generated columns.

```
select col, max(blocknum)
from stv_blocklist b, stv_tbl_perm p
where (b.tbl=p.id) and name = 'encodingvenue'
and col < 7
group by name, col
order by col;
```

The query returns the following results. The columns are numbered beginning with zero. Depending on how your cluster is configured, your result might have different numbers, but the relative sizes should be similar. You can see that BYTEDICT encoding on the second column produced the best results for this dataset. This approach has a compression ratio of better than 20:1. LZO and ZSTD encoding also produced excellent results. Different datasets produce different results, of course. When a column contains longer text strings, LZO often produces the best compression results.

col	max
0	203
1	10
2	22
3	204
4	56
5	72
6	20

(7 rows)

If you have data in an existing table, you can use the [ANALYZE COMPRESSION \(p. 665\)](#) command to view the suggested encodings for the table. For example, the following example shows the recommended encoding for a copy of the VENUE table, CARTESIAN_VENUE, that contains 38 million rows. Notice that ANALYZE COMPRESSION recommends LZO encoding for the VENUENAME column. ANALYZE COMPRESSION chooses optimal compression based on multiple factors, which include percent of reduction. In this specific case, BYTEDICT provides better compression, but LZO also produces greater than 90 percent compression.

```
analyze compression cartesian_venue;
```

Table	Column	Encoding	Est_reduction_pct
reallybigvenue	venueid	lzo	97.54
reallybigvenue	venuename	lzo	91.71
reallybigvenue	venuecity	lzo	96.01
reallybigvenue	venuestate	lzo	97.68
reallybigvenue	venueseats	lzo	98.21

Example: Choosing compression encodings for the CUSTOMER table

The following statement creates a CUSTOMER table that has columns with various data types. This CREATE TABLE statement shows one of many possible combinations of compression encodings for these columns.

```
create table customer(
    custkey int encode delta,
    custname varchar(30) encode raw,
    gender varchar(7) encode text255,
    address varchar(200) encode text255,
    city varchar(30) encode text255,
    state char(2) encode raw,
    zipcode char(5) encode bytedict,
    start_date date encode delta32k);
```

The following table shows the column encodings that were chosen for the CUSTOMER table and gives an explanation for the choices:

Column	Data type	Encoding	Explanation
CUSTKEY	int	delta	CUSTKEY consists of unique, consecutive integer values. Because the differences are one byte, DELTA is a good choice.
CUSTNAME	varchar(30)	raw	CUSTNAME has a large domain with few repeated values. Any compression encoding would probably be ineffective.
GENDER	varchar(7)	text255	GENDER is very small domain with many repeated values. Text255 works well with VARCHAR columns in which the same words recur.
ADDRESS	varchar(200)	text255	ADDRESS is a large domain, but contains many repeated words, such as Street, Avenue, North, South, and so on. Text 255 and text 32k are useful for compressing VARCHAR columns in which the same words recur. The column length is short, so text255 is a good choice.
CITY	varchar(30)	text255	CITY is a large domain, with some repeated values. Certain city names are used much more commonly than others. Text255 is a good choice for

Column	Data type	Encoding	Explanation
			the same reasons as ADDRESS.
STATE	char(2)	raw	In the United States, STATE is a precise domain of 50 two-character values. Bytedict encoding would yield some compression, but because the column size is only two characters, compression might not be worth the overhead of uncompressing the data.
ZIPCODE	char(5)	bytedict	ZIPCODE is a known domain of fewer than 50,000 unique values. Certain zip codes occur much more commonly than others. Bytedict encoding is very effective when a column contains a limited number of unique values.
START_DATE	date	delta32k	Delta encodings are very useful for date time columns, especially if the rows are loaded in date order.

Working with data distribution styles

When you load data into a table, Amazon Redshift distributes the rows of the table to each of the compute nodes according to the table's distribution style. When you run a query, the query optimizer redistributes the rows to the compute nodes as needed to perform any joins and aggregations. The goal in choosing a table distribution style is to minimize the impact of the redistribution step by locating the data where it must be before the query is run.

Note

This section will introduce you to the principles of data distribution in an Amazon Redshift database. We recommend that you create your tables with DISTSTYLE AUTO. If you do so, then Amazon Redshift uses automatic table optimization to choose the data distribution style. For more information, see [Working with automatic table optimization \(p. 36\)](#). The rest of this section provides details about distribution styles.

Topics

- [Data distribution concepts \(p. 50\)](#)

- [Distribution styles \(p. 51\)](#)
- [Viewing distribution styles \(p. 52\)](#)
- [Evaluating query patterns \(p. 53\)](#)
- [Designating distribution styles \(p. 53\)](#)
- [Evaluating the query plan \(p. 54\)](#)
- [Query plan example \(p. 55\)](#)
- [Distribution examples \(p. 58\)](#)

Data distribution concepts

Some data distribution concepts for Amazon Redshift follow.

Nodes and slices

An Amazon Redshift cluster is a set of nodes. Each node in the cluster has its own operating system, dedicated memory, and dedicated disk storage. One node is the *leader node*, which manages the distribution of data and query processing tasks to the compute nodes. The *compute nodes* provide resources to do those tasks.

The disk storage for a compute node is divided into a number of *slices*. The number of slices per node depends on the node size of the cluster. For example, each DS2.XL compute node has two slices, and each DS2.8XL compute node has 16 slices. The nodes all participate in running parallel queries, working on data that is distributed as evenly as possible across the slices. For more information about the number of slices that each node size has, see [About clusters and nodes](#) in the *Amazon Redshift Management Guide*.

Data redistribution

When you load data into a table, Amazon Redshift distributes the rows of the table to each of the node slices according to the table's distribution style. As part of a query plan, the optimizer determines where blocks of data must be located to best run the query. The data is then physically moved, or redistributed, while the query runs. Redistribution might involve either sending specific rows to nodes for joining or broadcasting an entire table to all of the nodes.

Data redistribution can account for a substantial portion of the cost of a query plan, and the network traffic it generates can affect other database operations and slow overall system performance. To the extent that you anticipate where best to locate data initially, you can minimize the impact of data redistribution.

Data distribution goals

When you load data into a table, Amazon Redshift distributes the table's rows to the compute nodes and slices according to the distribution style that you chose when you created the table. Data distribution has two primary goals:

- To distribute the workload uniformly among the nodes in the cluster. Uneven distribution, or data distribution skew, forces some nodes to do more work than others, which impairs query performance.
- To minimize data movement as a query runs. If the rows that participate in joins or aggregates are already collocated on the nodes with their joining rows in other tables, the optimizer doesn't need to redistribute as much data when queries run.

The distribution strategy that you choose for your database has important consequences for query performance, storage requirements, data loading, and maintenance. By choosing the best distribution style for each table, you can balance your data distribution and significantly improve overall system performance.

Distribution styles

When you create a table, you can designate one of four distribution styles; AUTO, EVEN, KEY, or ALL.

If you don't specify a distribution style, Amazon Redshift uses AUTO distribution.

AUTO distribution

With AUTO distribution, Amazon Redshift assigns an optimal distribution style based on the size of the table data. For example, if AUTO distribution style is specified, Amazon Redshift initially assigns the ALL distribution style to a small table. When the table grows larger, Amazon Redshift might change the distribution style to KEY, choosing the primary key (or a column of the composite primary key) as the distribution key. If the table grows larger and none of the columns are suitable to be the distribution key, Amazon Redshift changes the distribution style to EVEN. The change in distribution style occurs in the background with minimal impact to user queries.

To view actions that Amazon Redshift automatically performed to alter a table distribution key, see [SVL_AUTO_WORKER_ACTION \(p. 1629\)](#). To view current recommendations regarding altering a table distribution key, see [SVV.Alter_Table_Recommendations \(p. 1415\)](#).

To view the distribution style applied to a table, query the PG_CLASS_INFO system catalog view. For more information, see [Viewing distribution styles \(p. 52\)](#). If you don't specify a distribution style with the CREATE TABLE statement, Amazon Redshift applies AUTO distribution.

EVEN distribution

The leader node distributes the rows across the slices in a round-robin fashion, regardless of the values in any particular column. EVEN distribution is appropriate when a table doesn't participate in joins. It's also appropriate when there isn't a clear choice between KEY distribution and ALL distribution.

KEY distribution

The rows are distributed according to the values in one column. The leader node places matching values on the same node slice. If you distribute a pair of tables on the joining keys, the leader node collocates the rows on the slices according to the values in the joining columns. This way, matching values from the common columns are physically stored together.

ALL distribution

A copy of the entire table is distributed to every node. Where EVEN distribution or KEY distribution place only a portion of a table's rows on each node, ALL distribution ensures that every row is collocated for every join that the table participates in.

ALL distribution multiplies the storage required by the number of nodes in the cluster, and so it takes much longer to load, update, or insert data into multiple tables. ALL distribution is appropriate only for relatively slow moving tables; that is, tables that are not updated frequently or extensively. Because the cost of redistributing small tables during a query is low, there isn't a significant benefit to define small dimension tables as DISTSTYLE ALL.

Note

After you have specified a distribution style for a column, Amazon Redshift handles data distribution at the cluster level. Amazon Redshift does not require or support the concept of partitioning data within database objects. You don't need to create table spaces or define partitioning schemes for tables.

In certain scenarios, you can change the distribution style of a table after it is created. For more information, see [ALTER TABLE \(p. 640\)](#). For scenarios when you can't change the distribution style of a table after it's created, you can recreate the table and populate the new table with a deep copy. For more information, see [Performing a deep copy \(p. 102\)](#)

Viewing distribution styles

To view the distribution style of a table, query the PG_CLASS_INFO view or the SVV_TABLE_INFO view.

The RELEFFECTIVEDISTSTYLE column in PG_CLASS_INFO indicates the current distribution style for the table. If the table uses automatic distribution, RELEFFECTIVEDISTSTYLE is 10, 11, or 12, which indicates whether the effective distribution style is AUTO (ALL), AUTO (EVEN), or AUTO (KEY). If the table uses automatic distribution, the distribution style might initially show AUTO (ALL), then change to AUTO (EVEN) or AUTO (KEY) when the table grows.

The following table gives the distribution style for each value in RELEFFECTIVEDISTSTYLE column:

RELEFFECTIVEDISTSTYLE	Current distribution style
0	EVEN
1	KEY
8	ALL
10	AUTO (ALL)
11	AUTO (EVEN)
12	AUTO (KEY)

The DISTSTYLE column in SVV_TABLE_INFO indicates the current distribution style for the table. If the table uses automatic distribution, DISTSTYLE is AUTO (ALL), AUTO (EVEN), or AUTO (KEY).

The following example creates four tables using the three distribution styles and automatic distribution, then queries SVV_TABLE_INFO to view the distribution styles.

```

create table public.dist_key (col1 int)
diststyle key distkey (col1);

insert into public.dist_key values (1);

create table public.dist_even (col1 int)
diststyle even;

insert into public.dist_even values (1);

create table public.dist_all (col1 int)
diststyle all;

insert into public.dist_all values (1);

create table public.dist_auto (col1 int);

insert into public.dist_auto values (1);

select "schema", "table", diststyle from SVV_TABLE_INFO
where "table" like 'dist%';

      schema   |    table     | diststyle
-----+-----+-----+
  public  | dist_key   | KEY(col1)
  public  | dist_even  | EVEN
  public  | dist_all   | ALL

```

public	dist_auto	AUTO(ALL)
--------	-----------	-----------

Evaluating query patterns

Choosing distribution styles is only one aspect of database design. Consider distribution styles within the context of the entire system, balancing distribution with other important factors such as cluster size, compression encoding methods, sort keys, and table constraints.

Test your system with data that is as close to real data as possible.

To make good choices for distribution styles, you must understand the query patterns for your Amazon Redshift application. Identify the most costly queries in your system and base your initial database design on the demands of those queries. Factors that determine the total cost of a query include how long the query takes to run and how much computing resources it consumes. Other factors that determine query cost are how often it is run, and how disruptive it is to other queries and database operations.

Identify the tables that are used by the most costly queries, and evaluate their role in query runtime. Consider how the tables are joined and aggregated.

Use the guidelines in this section to choose a distribution style for each table. When you have done so, create the tables and load them with data that is as close as possible to real data. Then test the tables for the types of queries that you expect to use. You can evaluate the query explain plans to identify tuning opportunities. Compare load times, storage space, and query runtimes to balance your system's overall requirements.

Designating distribution styles

The considerations and recommendations for designating distribution styles in this section use a star schema as an example. Your database design might be based on a star schema, some variant of a star schema, or an entirely different schema. Amazon Redshift is designed to work effectively with whatever schema design you choose. The principles in this section can be applied to any design schema.

1. Specify the primary key and foreign keys for all your tables.

Amazon Redshift does not enforce primary key and foreign key constraints, but the query optimizer uses them when it generates query plans. If you set primary keys and foreign keys, your application must maintain the validity of the keys.

2. Distribute the fact table and its largest dimension table on their common columns.

Choose the largest dimension based on the size of dataset that participates in the most common join, not only the size of the table. If a table is commonly filtered, using a WHERE clause, only a portion of its rows participate in the join. Such a table has less impact on redistribution than a smaller table that contributes more data. Designate both the dimension table's primary key and the fact table's corresponding foreign key as DISTKEY. If multiple tables use the same distribution key, they are also collocated with the fact table. Your fact table can have only one distribution key. Any tables that join on another key isn't collocated with the fact table.

3. Designate distribution keys for the other dimension tables.

Distribute the tables on their primary keys or their foreign keys, depending on how they most commonly join with other tables.

4. Evaluate whether to change some of the dimension tables to use ALL distribution.

If a dimension table cannot be collocated with the fact table or other important joining tables, you can improve query performance significantly by distributing the entire table to all of the nodes. Using ALL distribution multiplies storage space requirements and increases load times and maintenance

operations, so you should weigh all factors before choosing ALL distribution. The following section explains how to identify candidates for ALL distribution by evaluating the EXPLAIN plan.

5. Use EVEN distribution for the remaining tables.

If a table is largely denormalized and does not participate in joins, or if you don't have a clear choice for another distribution style, use EVEN distribution.

To let Amazon Redshift choose the appropriate distribution style, don't explicitly specify a distribution style.

Evaluating the query plan

You can use query plans to identify candidates for optimizing the distribution style.

After making your initial design decisions, create your tables, load them with data, and test them. Use a test dataset that is as close as possible to the real data. Measure load times to use as a baseline for comparisons.

Evaluate queries that are representative of the most costly queries you expect to run, specifically queries that use joins and aggregations. Compare runtimes for various design options. When you compare runtimes, don't count the first time the query is run, because the first runtime includes the compilation time.

DS_DIST_NONE

No redistribution is required, because corresponding slices are colocated on the compute nodes. You typically have only one DS_DIST_NONE step, the join between the fact table and one dimension table.

DS_DIST_ALL_NONE

No redistribution is required, because the inner join table used DISTSTYLE ALL. The entire table is located on every node.

DS_DIST_INNER

The inner table is redistributed.

DS_DIST_OUTER

The outer table is redistributed.

DS_BCAST_INNER

A copy of the entire inner table is broadcast to all the compute nodes.

DS_DIST_ALL_INNER

The entire inner table is redistributed to a single slice because the outer table uses DISTSTYLE ALL.

DS_DIST_BOTH

Both tables are redistributed.

DS_DIST_NONE and DS_DIST_ALL_NONE are good. They indicate that no distribution was required for that step because all of the joins are colocated.

DS_DIST_INNER means that the step probably has a relatively high cost because the inner table is being redistributed to the nodes. DS_DIST_INNER indicates that the outer table is already properly distributed on the join key. Set the inner table's distribution key to the join key to convert this to DS_DIST_NONE. In some cases, distributing the inner table on the join key isn't possible because the outer table isn't distributed on the join key. If this is the case, evaluate whether to use ALL distribution for the inner table.

If the table isn't updated frequently or extensively, and it's large enough to carry a high redistribution cost, change the distribution style to ALL and test again. ALL distribution causes increased load times, so when you retest, include the load time in your evaluation factors.

DS_DIST_ALL_INNER is not good. It means that the entire inner table is redistributed to a single slice because the outer table uses DISTSTYLE ALL, so that a copy of the entire outer table is located on each node. This results in inefficient serial runtime of the join on a single node, instead taking advantage of parallel runtime using all of the nodes. DISTSTYLE ALL is meant to be used only for the inner join table. Instead, specify a distribution key or use even distribution for the outer table.

DS_BCAST_INNER and DS_DIST_BOTH are not good. Usually these redistributions occur because the tables are not joined on their distribution keys. If the fact table does not already have a distribution key, specify the joining column as the distribution key for both tables. If the fact table already has a distribution key on another column, evaluate whether changing the distribution key to collocate this join improve overall performance. If changing the distribution key of the outer table isn't an optimal choice, you can achieve collocation by specifying DISTSTYLE ALL for the inner table.

The following example shows a portion of a query plan with DS_BCAST_INNER and DS_DIST_NONE labels.

```
-> XN Hash Join DS_BCAST_INNER (cost=112.50..3272334142.59 rows=170771 width=84)
   Hash Cond: ("outer".venueid = "inner".venueid)
     -> XN Hash Join DS_BCAST_INNER (cost=109.98..3167290276.71 rows=172456 width=47)
        Hash Cond: ("outer".eventid = "inner".eventid)
          -> XN Merge Join DS_DIST_NONE (cost=0.00..6286.47 rows=172456 width=30)
             Merge Cond: ("outer".listid = "inner".listid)
               -> XN Seq Scan on listing (cost=0.00..1924.97 rows=192497 width=14)
               -> XN Seq Scan on sales (cost=0.00..1724.56 rows=172456 width=24)
```

After changing the dimension tables to use DISTSTYLE ALL, the query plan for the same query shows DS_DIST_ALL_NONE in place of DS_BCAST_INNER. Also, there is a dramatic change in the relative cost for the join steps.

```
-> XN Hash Join DS_DIST_ALL_NONE (cost=112.50..14142.59 rows=170771 width=84)
   Hash Cond: ("outer".venueid = "inner".venueid)
     -> XN Hash Join DS_DIST_ALL_NONE (cost=109.98..10276.71 rows=172456 width=47)
        Hash Cond: ("outer".eventid = "inner".eventid)
          -> XN Merge Join DS_DIST_NONE (cost=0.00..6286.47 rows=172456 width=30)
             Merge Cond: ("outer".listid = "inner".listid)
               -> XN Seq Scan on listing (cost=0.00..1924.97 rows=192497 width=14)
               -> XN Seq Scan on sales (cost=0.00..1724.56 rows=172456 width=24)
```

Query plan example

This example shows how to evaluate a query plan to find opportunities to optimize the distribution.

Run the following query with an EXPLAIN command to produce a query plan.

```
explain
select lastname, catname, venuename, venuecity, venuestate, eventname,
month, sum(pricepaid) as buyercost, max(totalprice) as maxtotalprice
from category join event on category.catid = event.catid
join venue on venue.venueid = event.venueid
join sales on sales.eventid = event.eventid
join listing on sales.listid = listing.listid
join date on sales.dateid = date.dateid
join users on users.userid = sales.buyerid
group by lastname, catname, venuename, venuecity, venuestate, eventname, month
```

```
having sum(pricepaid)>9999
order by catname, buyercost desc;
```

In the TICKIT database, SALES is a fact table and LISTING is its largest dimension. In order to collocate the tables, SALES is distributed on the LISTID, which is the foreign key for LISTING, and LISTING is distributed on its primary key, LISTID. The following example shows the CREATE TABLE commands for SALES and LISTING.

```
create table sales(
    salesid integer not null,
    listid integer not null distkey,
    sellerid integer not null,
    buyerid integer not null,
    eventid integer not null encode mostly16,
    dateid smallint not null,
    qtysold smallint not null encode mostly8,
    pricepaid decimal(8,2) encode delta32k,
    commission decimal(8,2) encode delta32k,
    saletime timestamp,
    primary key(salesid),
    foreign key(listid) references listing(listid),
    foreign key(sellerid) references users(userid),
    foreign key(buyerid) references users(userid),
    foreign key(dateid) references date(dateid)
        sortkey(listid,sellerid);

create table listing(
    listid integer not null distkey sortkey,
    sellerid integer not null,
    eventid integer not null encode mostly16,
    dateid smallint not null,
    numtickets smallint not null encode mostly8,
    priceperticket decimal(8,2) encode bytedict,
    totalprice decimal(8,2) encode mostly32,
    listtime timestamp,
    primary key(listid),
    foreign key(sellerid) references users(userid),
    foreign key(eventid) references event(eventid),
    foreign key(dateid) references date(dateid));
```

In the following query plan, the Merge Join step for the join on SALES and LISTING shows DS_DIST_NONE, which indicates that no redistribution is required for the step. However, moving up the query plan, the other inner joins show DS_BCAST_INNER, which indicates that the inner table is broadcast as part of the query execution. Because only one pair of tables can be collocated using key distribution, five tables must be rebroadcast.

```
QUERY PLAN
XN Merge  (cost=1015345167117.54..1015345167544.46 rows=1000 width=103)
  Merge Key: category.catname, sum(sales.pricepaid)
  -> XN Network  (cost=1015345167117.54..1015345167544.46 rows=170771 width=103)
      Send to leader
      -> XN Sort  (cost=1015345167117.54..1015345167544.46 rows=170771 width=103)
          Sort Key: category.catname, sum(sales.pricepaid)
          -> XN HashAggregate  (cost=15345150568.37..15345152276.08 rows=170771
width=103)
              Filter: (sum(pricepaid) > 9999.00)
              -> XN Hash Join DS_BCAST_INNER  (cost=742.08..15345146299.10
rows=170771 width=103)
                  Hash Cond: ("outer".catid = "inner".catid)
                  -> XN Hash Join DS_BCAST_INNER  (cost=741.94..15342942456.61
rows=170771 width=97)
                      Hash Cond: ("outer".dateid = "inner".dateid)
```

```

-> XN Hash Join DS_BCAST_INNER
(cost=737.38..15269938609.81 rows=170766 width=90)
    Hash Cond: ("outer".buyerid = "inner".userid)
        -> XN Hash Join DS_BCAST_INNER
(cost=112.50..3272334142.59 rows=170771 width=84)
    Hash Cond: ("outer".venueid = "inner".venueid)
        -> XN Hash Join DS_BCAST_INNER
(cost=109.98..3167290276.71 rows=172456 width=47)
    Hash Cond: ("outer".eventid =
"inner".eventid)
        -> XN Merge Join DS_DIST_NONE
(cost=0.00..6286.47 rows=172456 width=30)
    Merge Cond: ("outer".listid =
"inner".listid)
        -> XN Seq Scan on listing
(cost=0.00..1924.97 rows=192497 width=14)
        -> XN Seq Scan on sales
(cost=0.00..1724.56 rows=172456 width=24)
        -> XN Hash (cost=87.98..87.98
rows=8798 width=25)
        -> XN Seq Scan on event
(cost=0.00..87.98 rows=8798 width=25)
        -> XN Hash (cost=2.02..2.02 rows=202
width=41)
        -> XN Seq Scan on venue
(cost=0.00..2.02 rows=202 width=41)
        -> XN Hash (cost=499.90..499.90 rows=49990
width=14)
        -> XN Seq Scan on users (cost=0.00..499.90
rows=49990 width=14)
        -> XN Hash (cost=3.65..3.65 rows=365 width=11)
            -> XN Seq Scan on date (cost=0.00..3.65 rows=365
width=11)
        -> XN Hash (cost=0.11..0.11 rows=11 width=10)
            -> XN Seq Scan on category (cost=0.00..0.11 rows=11
width=10)

```

One solution is to alter the tables to have DISTSTYLE ALL.

```

ALTER TABLE users ALTER DISTSTYLE ALL;
ALTER TABLE venue ALTER DISTSTYLE ALL;
ALTER TABLE category ALTER DISTSTYLE ALL;
ALTER TABLE date ALTER DISTSTYLE ALL;
ALTER TABLE event ALTER DISTSTYLE ALL;

```

Run the same query with EXPLAIN again, and examine the new query plan. The joins now show DS_DIST_ALL_NONE, indicating that no redistribution is required because the data was distributed to every node using DISTSTYLE ALL.

```

QUERY PLAN
XN Merge (cost=1000000047117.54..1000000047544.46 rows=1000 width=103)
    Merge Key: category.catname, sum(sales.pricepaid)
        -> XN Network (cost=1000000047117.54..1000000047544.46 rows=170771 width=103)
            Send to leader
                -> XN Sort (cost=1000000047117.54..1000000047544.46 rows=170771 width=103)
                    Sort Key: category.catname, sum(sales.pricepaid)
                    -> XN HashAggregate (cost=30568.37..32276.08 rows=170771 width=103)
                        Filter: (sum(pricepaid) > 9999.00)
                        -> XN Hash Join DS_DIST_ALL_NONE (cost=742.08..26299.10 rows=170771
width=103)
                            Hash Cond: ("outer".buyerid = "inner".userid)
                                -> XN Hash Join DS_DIST_ALL_NONE (cost=117.20..21831.99
rows=170766 width=97)

```

```

Hash Cond: ("outer".dateid = "inner".dateid)
-> XN Hash Join DS_DIST_ALL_NONE (cost=112.64..17985.08
rows=170771 width=90)
          Hash Cond: ("outer".catid = "inner".catid)
          -> XN Hash Join DS_DIST_ALL_NONE
(cost=112.50..14142.59 rows=170771 width=84)
          Hash Cond: ("outer".venueid = "inner".venueid)
          -> XN Hash Join DS_DIST_ALL_NONE
(cost=109.98..10276.71 rows=172456 width=47)
          Hash Cond: ("outer".eventid =
"inner".eventid)
          -> XN Merge Join DS_DIST_NONE
(cost=0.00..6286.47 rows=172456 width=30)
          Merge Cond: ("outer".listid =
"inner".listid)
          -> XN Seq Scan on listing
(cost=0.00..1924.97 rows=192497 width=14)
          -> XN Seq Scan on sales
(cost=0.00..1724.56 rows=172456 width=24)
          -> XN Hash (cost=87.98..87.98 rows=8798
width=25)
          -> XN Seq Scan on event
(cost=0.00..87.98 rows=8798 width=25)
          -> XN Hash (cost=2.02..2.02 rows=202
width=41)
          -> XN Seq Scan on venue
(cost=0.00..2.02 rows=202 width=41)
          -> XN Hash (cost=0.11..0.11 rows=11 width=10)
          -> XN Seq Scan on category (cost=0.00..0.11
rows=11 width=10)
          -> XN Hash (cost=3.65..3.65 rows=365 width=11)
          -> XN Seq Scan on date (cost=0.00..3.65 rows=365
width=11)
          -> XN Hash (cost=499.90..499.90 rows=49990 width=14)
          -> XN Seq Scan on users (cost=0.00..499.90 rows=49990
width=14)

```

Distribution examples

The following examples show how data is distributed according to the options that you define in the CREATE TABLE statement.

DISTKEY examples

Look at the schema of the USERS table in the TICKIT database. USERID is defined as the SORTKEY column and the DISTKEY column:

column	type	encoding	distkey	sortkey
userid	integer	none	t	1
username	character(8)	none	f	0
firstname	character varying(30)	text32k	f	0
...				

USERID is a good choice for the distribution column on this table. If you query the SVV_DISKUSAGE system view, you can see that the table is very evenly distributed. Column numbers are zero-based, so USERID is column 0.

```
select slice, col, num_values as rows, minvalue, maxvalue
from svv_diskusage
where name='users' and col=0 and rows>0
order by slice, col;

slice| col | rows | minvalue | maxvalue
-----+---+-----+-----+
0 | 0 | 12496 | 4 | 49987
1 | 0 | 12498 | 1 | 49988
2 | 0 | 12497 | 2 | 49989
3 | 0 | 12499 | 3 | 49990
(4 rows)
```

The table contains 49,990 rows. The rows (num_values) column shows that each slice contains about the same number of rows. The minvalue and maxvalue columns show the range of values on each slice. Each slice includes nearly the entire range of values, so there's a good chance that every slice participates in running a query that filters for a range of user IDs.

This example demonstrates distribution on a small test system. The total number of slices is typically much higher.

If you commonly join or group using the STATE column, you might choose to distribute on the STATE column. The following example shows a case where you create a new table with the same data as the USERS table but set the DISTKEY to the STATE column. In this case, the distribution isn't as even. Slice 0 (13,587 rows) holds approximately 30 percent more rows than slice 3 (10,150 rows). In a much larger table, this amount of distribution skew can have an adverse impact on query processing.

```
create table userskey distkey(state) as select * from users;

select slice, col, num_values as rows, minvalue, maxvalue from svv_diskusage
where name = 'userskey' and col=0 and rows>0
order by slice, col;

slice | col | rows | minvalue | maxvalue
-----+---+-----+-----+
0 | 0 | 13587 | 5 | 49989
1 | 0 | 11245 | 2 | 49990
2 | 0 | 15008 | 1 | 49976
3 | 0 | 10150 | 4 | 49986
(4 rows)
```

DISTSTYLE EVEN example

If you create a new table with the same data as the USERS table but set the DISTSTYLE to EVEN, rows are always evenly distributed across slices.

```
create table userseven diststyle even as
select * from users;

select slice, col, num_values as rows, minvalue, maxvalue from svv_diskusage
where name = 'userseven' and col=0 and rows>0
order by slice, col;

slice | col | rows | minvalue | maxvalue
-----+---+-----+-----+
0 | 0 | 12497 | 4 | 49990
1 | 0 | 12498 | 8 | 49984
2 | 0 | 12498 | 2 | 49988
3 | 0 | 12497 | 1 | 49989
(4 rows)
```

However, because distribution is not based on a specific column, query processing can be degraded, especially if the table is joined to other tables. The lack of distribution on a joining column often influences the type of join operation that can be performed efficiently. Joins, aggregations, and grouping operations are optimized when both tables are distributed and sorted on their respective joining columns.

DISTSTYLE ALL example

If you create a new table with the same data as the USERS table but set the DISTSTYLE to ALL, all the rows are distributed to the first slice of each node.

```
select slice, col, num_values as rows, minvalue, maxvalue from svv_diskusage
where name = 'usersall' and col=0 and rows > 0
order by slice, col;

slice | col | rows | minvalue | maxvalue
-----+-----+-----+-----+
  0 |   0 | 49990 |        4 |    49990
  2 |   0 | 49990 |        2 |    49990

(4 rows)
```

Working with sort keys

Note

We recommend that you create your tables with SORTKEY AUTO. If you do so, then Amazon Redshift uses automatic table optimization to choose the sort key. For more information, see [Working with automatic table optimization \(p. 36\)](#). The rest of this section provides details about the sort order.

When you create a table, you can alternatively define one or more of its columns as *sort keys*. When data is initially loaded into the empty table, the rows are stored on disk in sorted order. Information about sort key columns is passed to the query planner, and the planner uses this information to construct plans that exploit the way that the data is sorted. For more information, see [CREATE TABLE \(p. 823\)](#). For information on best practices when creating a sort key, see [Choose the best sort key \(p. 15\)](#).

Sorting enables efficient handling of range-restricted predicates. Amazon Redshift stores columnar data in 1 MB disk blocks. The min and max values for each block are stored as part of the metadata. If a query uses a range-restricted predicate, the query processor can use the min and max values to rapidly skip over large numbers of blocks during table scans. For example, suppose that a table stores five years of data sorted by date and a query specifies a date range of one month. In this case, you can remove up to 98 percent of the disk blocks from the scan. If the data is not sorted, more of the disk blocks (possibly all of them) have to be scanned.

You can specify either a compound or interleaved sort key. A compound sort key is more efficient when query predicates use a *prefix*, which is a subset of the sort key columns in order. An interleaved sort key gives equal weight to each column in the sort key, so query predicates can use any subset of the columns that make up the sort key, in any order.

To understand the impact of the chosen sort key on query performance, use the [EXPLAIN \(p. 881\)](#) command. For more information, see [Query planning and execution workflow \(p. 463\)](#).

To define a sort type, use either the INTERLEAVED or COMPOUND keyword with your CREATE TABLE or CREATE TABLE AS statement. The default is COMPOUND. COMPOUND is recommended when you update your table regularly with INSERT, UPDATE, or DELETE operations. An INTERLEAVED sort key can use a maximum of eight columns. Depending on your data and cluster size, VACUUM REINDEX takes significantly longer than VACUUM FULL because it makes an additional pass to analyze the interleaved

sort keys. The sort and merge operation can take longer for interleaved tables because the interleaved sort might have to rearrange more rows than a compound sort.

To view the sort keys for a table, query the [SVV_TABLE_INFO \(p. 1457\)](#) system view.

Topics

- [Compound sort key \(p. 61\)](#)
- [Interleaved sort key \(p. 61\)](#)

Compound sort key

A compound key is made up of all of the columns listed in the sort key definition, in the order they are listed. A compound sort key is most useful when a query's filter applies conditions, such as filters and joins, that use a prefix of the sort keys. The performance benefits of compound sorting decrease when queries depend only on secondary sort columns, without referencing the primary columns. COMPOUND is the default sort type.

Compound sort keys might speed up joins, GROUP BY and ORDER BY operations, and window functions that use PARTITION BY and ORDER BY. For example, a merge join, which is often faster than a hash join, is feasible when the data is distributed and presorted on the joining columns. Compound sort keys also help improve compression.

As you add rows to a sorted table that already contains data, the unsorted region grows, which has a significant effect on performance. The effect is greater when the table uses interleaved sorting, especially when the sort columns include data that increases monotonically, such as date or timestamp columns. Run a VACUUM operation regularly, especially after large data loads, to re-sort and re-analyze the data. For more information, see [Managing the size of the unsorted region \(p. 113\)](#). After vacuuming to resort the data, it's a good practice to run an ANALYZE command to update the statistical metadata for the query planner. For more information, see [Analyzing tables \(p. 105\)](#).

Interleaved sort key

An interleaved sort gives equal weight to each column, or subset of columns, in the sort key. If multiple queries use different columns for filters, then you can often improve performance for those queries by using an interleaved sort style. When a query uses restrictive predicates on secondary sort columns, interleaved sorting significantly improves query performance as compared to compound sorting.

Important

Don't use an interleaved sort key on columns with monotonically increasing attributes, such as identity columns, dates, or timestamps.

The performance improvements you gain by implementing an interleaved sort key should be weighed against increased load and vacuum times.

Interleaved sorts are most effective with highly selective queries that filter on one or more of the sort key columns in the WHERE clause, for example `select c_name from customer where c_region = 'ASIA'`. The benefits of interleaved sorting increase with the number of sorted columns that are restricted.

An interleaved sort is more effective with large tables. Sorting is applied on each slice. Thus, an interleaved sort is most effective when a table is large enough to require multiple 1 MB blocks per slice. Here, the query processor can skip a significant proportion of the blocks using restrictive predicates. To view the number of blocks a table uses, query the [STV_BLOCKLIST \(p. 1579\)](#) system view.

When sorting on a single column, an interleaved sort might give better performance than a compound sort if the column values have a long common prefix. For example, URLs commonly begin with "http://www". Compound sort keys use a limited number of characters from the prefix, which results in a lot

of duplication of keys. Interleaved sorts use an internal compression scheme for zone map values that enables them to better discriminate among column values that have a long common prefix.

When migrating Amazon Redshift provisioned clusters to Amazon Redshift Serverless, Redshift converts tables with interleaved sort keys and DISTSTYLE KEY to compound sort keys. The DISTSTYLE doesn't change. For more information on distribution styles, see [Working with data distribution styles](#).

VACUUM REINDEX

As you add rows to a sorted table that already contains data, performance might deteriorate over time. This deterioration occurs for both compound and interleaved sorts, but it has a greater effect on interleaved tables. A VACUUM restores the sort order, but the operation can take longer for interleaved tables because merging new interleaved data might involve modifying every data block.

When tables are initially loaded, Amazon Redshift analyzes the distribution of the values in the sort key columns and uses that information for optimal interleaving of the sort key columns. As a table grows, the distribution of the values in the sort key columns can change, or skew, especially with date or timestamp columns. If the skew becomes too large, performance might be affected. To re-analyze the sort keys and restore performance, run the VACUUM command with the REINDEX key word. Because it must take an extra analysis pass over the data, VACUUM REINDEX can take longer than a standard VACUUM for interleaved tables. To view information about key distribution skew and last reindex time, query the [SVV_INTERLEAVED_COLUMNS \(p. 1434\)](#) system view.

For more information about how to determine how often to run VACUUM and when to run a VACUUM REINDEX, see [Deciding whether to reindex \(p. 112\)](#).

Defining table constraints

Uniqueness, primary key, and foreign key constraints are informational only; *they are not enforced by Amazon Redshift* when you populate a table. For example, if you insert data into a table with dependencies, the insert can succeed even if it violates the constraint. Nonetheless, primary keys and foreign keys are used as planning hints and they should be declared if your ETL process or some other process in your application enforces their integrity.

For example, the query planner uses primary and foreign keys in certain statistical computations. It does this to infer uniqueness and referential relationships that affect subquery decorrelation techniques. By doing this, it can order large numbers of joins and remove redundant joins.

The planner leverages these key relationships, but it assumes that all keys in Amazon Redshift tables are valid as loaded. If your application allows invalid foreign keys or primary keys, some queries could return incorrect results. For example, a SELECT DISTINCT query might return duplicate rows if the primary key is not unique. Do not define key constraints for your tables if you doubt their validity. However, always declare primary and foreign keys and uniqueness constraints when you know that they are valid.

Amazon Redshift *does* enforce NOT NULL column constraints.

For more information about table constraints, see [CREATE TABLE \(p. 823\)](#). For information about how to drop a table with dependencies, see [DROP TABLE \(p. 874\)](#).

Loading data

Topics

- [Using a COPY command to load data \(p. 63\)](#)
- [Continuous file ingestion from Amazon S3 \(preview\) \(p. 94\)](#)
- [Updating tables with DML commands \(p. 96\)](#)
- [Updating and inserting new data \(p. 96\)](#)
- [Performing a deep copy \(p. 102\)](#)
- [Analyzing tables \(p. 105\)](#)
- [Vacuuming tables \(p. 110\)](#)
- [Managing concurrent write operations \(p. 118\)](#)
- [Tutorial: Loading data from Amazon S3 \(p. 124\)](#)

A COPY command is the most efficient way to load a table. You can also add data to your tables using INSERT commands, though it is much less efficient than using COPY. The COPY command is able to read from multiple data files or multiple data streams simultaneously. Amazon Redshift allocates the workload to the cluster nodes and performs the load operations in parallel, including sorting the rows and distributing data across node slices.

Note

Amazon Redshift Spectrum external tables are read-only. You can't COPY or INSERT to an external table.

To access data on other AWS resources, your cluster must have permission to access those resources and to perform the necessary actions to access the data. You can use AWS Identity and Access Management (IAM) to limit the access users have to your cluster resources and data.

After your initial data load, if you add, modify, or delete a significant amount of data, you should follow up by running a VACUUM command to reorganize your data and reclaim space after deletes. You should also run an ANALYZE command to update table statistics.

This section explains how to load data and troubleshoot data loads and presents best practices for loading data.

Using a COPY command to load data

Topics

- [Credentials and access permissions \(p. 64\)](#)
- [Preparing your input data \(p. 65\)](#)
- [Loading data from Amazon S3 \(p. 66\)](#)
- [Loading data from Amazon EMR \(p. 74\)](#)
- [Loading data from remote hosts \(p. 78\)](#)
- [Loading data from an Amazon DynamoDB table \(p. 84\)](#)
- [Verifying that the data loaded correctly \(p. 86\)](#)
- [Validating input data \(p. 86\)](#)
- [Loading tables with automatic compression \(p. 87\)](#)
- [Optimizing storage for narrow tables \(p. 89\)](#)
- [Loading default column values \(p. 89\)](#)
- [Troubleshooting data loads \(p. 90\)](#)

The COPY command leverages the Amazon Redshift massively parallel processing (MPP) architecture to read and load data in parallel from files on Amazon S3, from a DynamoDB table, or from text output from one or more remote hosts.

Note

We strongly recommend using the COPY command to load large amounts of data. Using individual INSERT statements to populate a table might be prohibitively slow. Alternatively, if your data already exists in other Amazon Redshift database tables, use INSERT INTO ... SELECT or CREATE TABLE AS to improve performance. For information, see [INSERT \(p. 902\)](#) or [CREATE TABLE AS \(p. 838\)](#).

To load data from another AWS resource, your cluster must have permission to access the resource and perform the necessary actions.

To grant or revoke privilege to load data into a table using a COPY command, grant or revoke the INSERT privilege.

Your data needs to be in the proper format for loading into your Amazon Redshift table. This section presents guidelines for preparing and verifying your data before the load and for validating a COPY statement before you run it.

To protect the information in your files, you can encrypt the data files before you upload them to your Amazon S3 bucket; COPY will decrypt the data as it performs the load. You can also limit access to your load data by providing temporary security credentials to users. Temporary security credentials provide enhanced security because they have short life spans and cannot be reused after they expire.

Amazon Redshift has features built in to COPY to load uncompressed, delimited data quickly. But you can compress your files using gzip, lzop, or bzip2 to save time uploading the files.

If the following keywords are in the COPY query, automatic splitting of uncompressed data is not supported: ESCAPE, REMOVEQUOTES, and FIXEDWIDTH. But the CSV keyword is supported.

To help keep your data secure in transit within the AWS Cloud, Amazon Redshift uses hardware accelerated SSL to communicate with Amazon S3 or Amazon DynamoDB for COPY, UNLOAD, backup, and restore operations.

When you load your table directly from an Amazon DynamoDB table, you have the option to control the amount of Amazon DynamoDB provisioned throughput you consume.

You can optionally let COPY analyze your input data and automatically apply optimal compression encodings to your table as part of the load process.

Credentials and access permissions

To load or unload data using another AWS resource, such as Amazon S3, Amazon DynamoDB, Amazon EMR, or Amazon EC2, your cluster must have permission to access the resource and perform the necessary actions to access the data. For example, to load data from Amazon S3, COPY must have LIST access to the bucket and GET access for the bucket objects.

To obtain authorization to access a resource, your cluster must be authenticated. You can choose either role-based access control or key-based access control. This section presents an overview of the two methods. For complete details and examples, see [Permissions to access other AWS Resources \(p. 714\)](#).

Role-based access control

With role-based access control, your cluster temporarily assumes an AWS Identity and Access Management (IAM) role on your behalf. Then, based on the authorizations granted to the role, your cluster can access the required AWS resources.

We recommend using role-based access control because it provides more secure, fine-grained control of access to AWS resources and sensitive user data, in addition to safeguarding your AWS credentials.

To use role-based access control, you must first create an IAM role using the Amazon Redshift service role type, and then attach the role to your cluster. The role must have, at a minimum, the permissions listed in [IAM permissions for COPY, UNLOAD, and CREATE LIBRARY \(p. 718\)](#). For steps to create an IAM role and attach it to your cluster, see [Creating an IAM Role to Allow Your Amazon Redshift Cluster to Access AWS Services](#) in the *Amazon Redshift Management Guide*.

You can add a role to a cluster or view the roles associated with a cluster by using the Amazon Redshift Management Console, CLI, or API. For more information, see [Authorizing COPY and UNLOAD Operations Using IAM Roles](#) in the *Amazon Redshift Management Guide*.

When you create an IAM role, IAM returns an Amazon Resource Name (ARN) for the role. To run a COPY command using an IAM role, provide the role ARN using the IAM_ROLE parameter or the CREDENTIALS parameter.

The following COPY command example uses IAM_ROLE parameter with the role MyRedshiftRole for authentication.

```
copy customer from 's3://mybucket/mydata'  
iam_role 'arn:aws:iam::12345678901:role/MyRedshiftRole';
```

The AWS user must have, at a minimum, the permissions listed in [IAM permissions for COPY, UNLOAD, and CREATE LIBRARY \(p. 718\)](#).

Key-based access control

With key-based access control, you provide the access key ID and secret access key for a user that is authorized to access the AWS resources that contain the data.

Note

We strongly recommend using an IAM role for authentication instead of supplying a plain-text access key ID and secret access key. If you choose key-based access control, never use your AWS account (root) credentials. Always create an IAM user and provide that user's access key ID and secret access key. For steps to create an IAM user, see [Creating an IAM User in Your AWS Account](#).

Preparing your input data

If your input data is not compatible with the table columns that will receive it, the COPY command will fail.

Use the following guidelines to help ensure that your input data is valid:

- Your data can only contain UTF-8 characters up to four bytes long.
- Verify that CHAR and VARCHAR strings are no longer than the lengths of the corresponding columns. VARCHAR strings are measured in bytes, not characters, so, for example, a four-character string of Chinese characters that occupy four bytes each requires a VARCHAR(16) column.
- Multibyte characters can only be used with VARCHAR columns. Verify that multibyte characters are no more than four bytes long.
- Verify that data for CHAR columns only contains single-byte characters.
- Do not include any special characters or syntax to indicate the last field in a record. This field can be a delimiter.
- If your data includes null terminators, also referred to as NUL (UTF-8 0000) or binary zero (0x000), you can load these characters as NULLS into CHAR or VARCHAR columns by using the NULL AS option in the COPY command: null as '\0' or null as '\000'. If you do not use NULL AS, null terminators will cause your COPY to fail.

- If your strings contain special characters, such as delimiters and embedded newlines, use the ESCAPE option with the [COPY \(p. 677\)](#) command.
- Verify that all single and double quotation marks are appropriately matched.
- Verify that floating-point strings are in either standard floating-point format, such as 12.123, or an exponential format, such as 1.0E4.
- Verify that all timestamp and date strings follow the specifications for [DATEFORMAT and TIMEFORMAT strings \(p. 724\)](#). The default timestamp format is YYYY-MM-DD hh:mm:ss, and the default date format is YYYY-MM-DD.
- For more information about boundaries and limitations on individual data types, see [Data types \(p. 574\)](#). For information about multibyte character errors, see [Multibyte character load errors \(p. 92\)](#)

Loading data from Amazon S3

Topics

- [Loading data from compressed and uncompressed files \(p. 66\)](#)
- [Uploading files to Amazon S3 \(p. 67\)](#)
- [Using the COPY command to load from Amazon S3 \(p. 70\)](#)

The COPY command leverages the Amazon Redshift massively parallel processing (MPP) architecture to read and load data in parallel from a file or multiple files in an Amazon S3 bucket. You can take maximum advantage of parallel processing by splitting your data into multiple files, in cases where the files are compressed. (There are exceptions to this rule. These are detailed in [Loading data files](#).) You can also take maximum advantage of parallel processing by setting distribution keys on your tables. For more information about distribution keys, see [Working with data distribution styles \(p. 49\)](#).

Data is loaded into the target table, one line per row. The fields in the data file are matched to table columns in order, left to right. Fields in the data files can be fixed-width or character delimited; the default delimiter is a pipe (|). By default, all the table columns are loaded, but you can optionally define a comma-separated list of columns. If a table column is not included in the column list specified in the COPY command, it is loaded with a default value. For more information, see [Loading default column values \(p. 89\)](#).

Loading data from compressed and uncompressed files

When you load compressed data, we recommend that you split the data for each table into multiple files. When you load uncompressed, delimited data, the COPY command uses massively parallel processing (MPP) and scan ranges to load data from large files in an Amazon S3 bucket.

Loading data from multiple compressed files

In cases where you have compressed data, we recommend that you split the data for each table into multiple files. The COPY command can load data from multiple files in parallel. You can load multiple files by specifying a common prefix, or *prefix key*, for the set, or by explicitly listing the files in a manifest file.

Split your data into files so that the number of files is a multiple of the number of slices in your cluster. That way, Amazon Redshift can divide the data evenly among the slices. The number of slices per node depends on the node size of the cluster. For example, each ds2.xl compute node has two slices, and each ds2.8xl compute node has 32 slices. For more information about the number of slices that each node size has, see [About clusters and nodes](#) in the *Amazon Redshift Management Guide*.

The nodes all participate in running parallel queries, working on data that is distributed as evenly as possible across the slices. If you have a cluster with two ds2.xl nodes, you might split your data into four

files or some multiple of four. Amazon Redshift doesn't take file size into account when dividing the workload. Thus, you need to ensure that the files are roughly the same size, from 1 MB to 1 GB after compression.

To use object prefixes to identify the load files, name each file with a common prefix. For example, you might split the `venue.txt` file into four files, as follows.

```
venue.txt.1
venue.txt.2
venue.txt.3
venue.txt.4
```

If you put multiple files in a folder in your bucket and specify the folder name as the prefix, COPY loads all of the files in the folder. If you explicitly list the files to be loaded by using a manifest file, the files can reside in different buckets or folders.

For more information about manifest files, see [Example: COPY from Amazon S3 using a manifest \(p. 727\)](#).

Loading data from uncompressed, delimited files

When you load uncompressed, delimited data, the COPY command uses the massively parallel processing (MPP) architecture in Amazon Redshift. Amazon Redshift automatically uses slices working in parallel to load ranges of data from a large file in an Amazon S3 bucket. The file must be delimited for parallel loading to occur. For example, pipe delimited. Automatic, parallel data loading with the COPY command is also available for CSV files. You can also take advantage of parallel processing by setting distribution keys on your tables. For more information about distribution keys, see [Working with data distribution styles \(p. 49\)](#).

Automatic, parallel data loading isn't supported when the COPY query includes any of the following keywords: ESCAPE, REMOVEQUOTES, and FIXEDWIDTH.

Data from the file or files is loaded into the target table, one line per row. The fields in the data file are matched to table columns in order, left to right. Fields in the data files can be fixed-width or character delimited; the default delimiter is a pipe (|). By default, all the table columns are loaded, but you can optionally define a comma-separated list of columns. If a table column isn't included in the column list specified in the COPY command, it's loaded with a default value. For more information, see [Loading default column values \(p. 89\)](#).

Follow this general process to load data from Amazon S3, when your data is uncompressed and delimited:

1. Upload your files to Amazon S3.
2. Run a COPY command to load the table.
3. Verify that the data was loaded correctly.

For examples of COPY commands, see [COPY examples \(p. 726\)](#). For information about data loaded into Amazon Redshift, check the [STL_LOAD_COMMITS \(p. 1523\)](#) and [STL_LOAD_ERRORS \(p. 1526\)](#) system tables.

For more information about nodes and the slices contained in each, see [About clusters and nodes](#) in the *Amazon Redshift Management Guide*.

Uploading files to Amazon S3

Topics

- [Managing data consistency \(p. 68\)](#)

- [Uploading encrypted data to Amazon S3 \(p. 68\)](#)
- [Verifying that the correct files are present in your bucket \(p. 69\)](#)

There are a couple approaches to take when uploading text files to Amazon S3:

- If you have compressed files, we recommend that you split large files to take advantage of parallel processing in Amazon Redshift.
- On the other hand, COPY automatically splits large, uncompressed, text-delimited file data to facilitate parallelism and effectively distribute the data from large files.

Create an Amazon S3 bucket to hold your data files, and then upload the data files to the bucket. For information about creating buckets and uploading files, see [Working with Amazon S3 Buckets](#) in the *Amazon Simple Storage Service User Guide*.

Important

The Amazon S3 bucket that holds the data files must be created in the same AWS Region as your cluster unless you use the [REGION \(p. 686\)](#) option to specify the Region in which the Amazon S3 bucket is located.

Ensure that the S3 IP ranges are added to your allowlist. To learn more about the required S3 IP ranges, see [Network isolation](#).

You can create an Amazon S3 bucket in a specific Region either by selecting the Region when you create the bucket by using the Amazon S3 console, or by specifying an endpoint when you create the bucket using the Amazon S3 API or CLI.

Following the data load, verify that the correct files are present on Amazon S3.

Managing data consistency

Amazon S3 provides strong read-after-write consistency for COPY, UNLOAD, INSERT (external table), CREATE EXTERNAL TABLE AS, and Amazon Redshift Spectrum operations on Amazon S3 buckets in all AWS Regions. In addition, read operations on Amazon S3 Select, Amazon S3 Access Control Lists, Amazon S3 Object Tags, and object metadata (for example, HEAD object) are strongly consistent. For more information about data consistency, see [Amazon S3 Data Consistency Model](#) in the *Amazon Simple Storage Service User Guide*.

Uploading encrypted data to Amazon S3

Amazon S3 supports both server-side encryption and client-side encryption. This topic discusses the differences between the server-side and client-side encryption and describes the steps to use client-side encryption with Amazon Redshift. Server-side encryption is transparent to Amazon Redshift.

Server-side encryption

Server-side encryption is data encryption at rest—that is, Amazon S3 encrypts your data as it uploads it and decrypts it for you when you access it. When you load tables using a COPY command, there is no difference in the way you load from server-side encrypted or unencrypted objects on Amazon S3. For more information about server-side encryption, see [Using Server-Side Encryption](#) in the *Amazon Simple Storage Service User Guide*.

Client-side encryption

In client-side encryption, your client application manages encryption of your data, the encryption keys, and related tools. You can upload data to an Amazon S3 bucket using client-side encryption, and then load the data using the COPY command with the ENCRYPTED option and a private encryption key to provide greater security.

You encrypt your data using envelope encryption. With *envelope encryption*, your application handles all encryption exclusively. Your private encryption keys and your unencrypted data are never sent to AWS, so it's very important that you safely manage your encryption keys. If you lose your encryption keys, you won't be able to unencrypt your data, and you can't recover your encryption keys from AWS. Envelope encryption combines the performance of fast symmetric encryption while maintaining the greater security that key management with asymmetric keys provides. A one-time-use symmetric key (the envelope symmetric key) is generated by your Amazon S3 encryption client to encrypt your data, then that key is encrypted by your root key and stored alongside your data in Amazon S3. When Amazon Redshift accesses your data during a load, the encrypted symmetric key is retrieved and decrypted with your real key, then the data is decrypted.

To work with Amazon S3 client-side encrypted data in Amazon Redshift, follow the steps outlined in [Protecting Data Using Client-Side Encryption](#) in the *Amazon Simple Storage Service User Guide*, with the additional requirements that you use:

- **Symmetric encryption** – The AWS SDK for Java AmazonS3EncryptionClient class uses envelope encryption, described preceding, which is based on symmetric key encryption. Use this class to create an Amazon S3 client to upload client-side encrypted data.
- **A 256-bit AES root symmetric key** – A root key encrypts the envelope key. You pass the root key to your instance of the AmazonS3EncryptionClient class. Save this key, because you will need it to copy data into Amazon Redshift.
- **Object metadata to store encrypted envelope key** – By default, Amazon S3 stores the envelope key as object metadata for the AmazonS3EncryptionClient class. The encrypted envelope key that is stored as object metadata is used during the decryption process.

Note

If you get a cipher encryption error message when you use the encryption API for the first time, your version of the JDK may have a Java Cryptography Extension (JCE) jurisdiction policy file that limits the maximum key length for encryption and decryption transformations to 128 bits. For information about addressing this issue, go to [Specifying Client-Side Encryption Using the AWS SDK for Java](#) in the *Amazon Simple Storage Service User Guide*.

For information about loading client-side encrypted files into your Amazon Redshift tables using the COPY command, see [Loading encrypted data files from Amazon S3 \(p. 73\)](#).

Example: Uploading client-side encrypted data

For an example of how to use the AWS SDK for Java to upload client-side encrypted data, go to [Protecting data using client-side encryption](#) in the *Amazon Simple Storage Service User Guide*.

The second option shows the choices you must make during client-side encryption so that the data can be loaded in Amazon Redshift. Specifically, the example shows using object metadata to store the encrypted envelope key and the use of a 256-bit AES root symmetric key.

This example provides example code using the AWS SDK for Java to create a 256-bit AES symmetric root key and save it to a file. Then the example upload an object to Amazon S3 using an S3 encryption client that first encrypts sample data on the client-side. The example also downloads the object and verifies that the data is the same.

Verifying that the correct files are present in your bucket

After you upload your files to your Amazon S3 bucket, we recommend listing the contents of the bucket to verify that all of the correct files are present and that no unwanted files are present. For example, if the bucket mybucket holds a file named venue.txt.back, that file will be loaded, perhaps unintentionally, by the following command:

```
copy venue from 's3://mybucket/venue' ... ;
```

If you want to control specifically which files are loaded, you can use a manifest file to explicitly list the data files. For more information about using a manifest file, see the [copy_from_s3_manifest_file \(p. 685\)](#) option for the COPY command and [Example: COPY from Amazon S3 using a manifest \(p. 727\)](#) in the COPY examples.

For more information about listing the contents of the bucket, see [Listing Object Keys](#) in the *Amazon S3 Developer Guide*.

Using the COPY command to load from Amazon S3

Topics

- [Using a manifest to specify data files \(p. 71\)](#)
- [Loading compressed data files from Amazon S3 \(p. 72\)](#)
- [Loading fixed-width data from Amazon S3 \(p. 72\)](#)
- [Loading multibyte data from Amazon S3 \(p. 73\)](#)
- [Loading encrypted data files from Amazon S3 \(p. 73\)](#)

Use the [COPY \(p. 677\)](#) command to load a table in parallel from data files on Amazon S3. You can specify the files to be loaded by using an Amazon S3 object prefix or by using a manifest file.

The syntax to specify the files to be loaded by using a prefix is as follows:

```
copy <table_name> from 's3://<bucket_name>/<object_prefix>'  
authorization;
```

The manifest file is a JSON-formatted file that lists the data files to be loaded. The syntax to specify the files to be loaded by using a manifest file is as follows:

```
copy <table_name> from 's3://<bucket_name>/<manifest_file>'  
authorization  
manifest;
```

The table to be loaded must already exist in the database. For information about creating a table, see [CREATE TABLE \(p. 823\)](#) in the SQL Reference.

The values for *authorization* provide the AWS authorization your cluster needs to access the Amazon S3 objects. For information about required permissions, see [IAM permissions for COPY, UNLOAD, and CREATE LIBRARY \(p. 718\)](#). The preferred method for authentication is to specify the IAM_ROLE parameter and provide the Amazon Resource Name (ARN) for an IAM role with the necessary permissions. For more information, see [Role-based access control \(p. 715\)](#).

To authenticate using the IAM_ROLE parameter, replace `<aws-account-id>` and `<role-name>` as shown in the following syntax.

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

The following example shows authentication using an IAM role.

```
copy customer  
from 's3://mybucket/mydata'  
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

For more information about other authorization options, see [Authorization parameters \(p. 693\)](#)

If you want to validate your data without actually loading the table, use the NOLOAD option with the [COPY \(p. 677\)](#) command.

The following example shows the first few rows of a pipe-delimited data in a file named `venue.txt`.

```
1|Toyota Park|Bridgeview|IL|0
2|Columbus Crew Stadium|Columbus|OH|0
3|RFK Stadium|Washington|DC|0
```

Before uploading the file to Amazon S3, split the file into multiple files so that the `COPY` command can load it using parallel processing. The number of files should be a multiple of the number of slices in your cluster. Split your load data files so that the files are about equal size, between 1 MB and 1 GB after compression. For more information, see [Loading data from compressed and uncompressed files \(p. 66\)](#).

For example, the `venue.txt` file might be split into four files, as follows:

```
venue.txt.1
venue.txt.2
venue.txt.3
venue.txt.4
```

The following `COPY` command loads the `VENUE` table using the pipe-delimited data in the data files with the prefix 'venue' in the Amazon S3 bucket `mybucket`.

Note

The Amazon S3 bucket `mybucket` in the following examples does not exist. For sample `COPY` commands that use real data in an existing Amazon S3 bucket, see [Load sample data](#).

```
copy venue from 's3://mybucket/venue'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

If no Amazon S3 objects with the key prefix 'venue' exist, the load fails.

Using a manifest to specify data files

You can use a manifest to make sure that the `COPY` command loads all of the required files, and only the required files, for a data load. You can use a manifest to load files from different buckets or files that do not share the same prefix. Instead of supplying an object path for the `COPY` command, you supply the name of a JSON-formatted text file that explicitly lists the files to be loaded. The URL in the manifest must specify the bucket name and full object path for the file, not just a prefix.

For more information about manifest files, see the `COPY` example [Using a manifest to specify data files \(p. 727\)](#).

The following example shows the JSON to load files from different buckets and with file names that begin with date stamps.

```
{
  "entries": [
    {"url": "s3://mybucket-alpha/2013-10-04-custdata", "mandatory": true},
    {"url": "s3://mybucket-alpha/2013-10-05-custdata", "mandatory": true},
    {"url": "s3://mybucket-beta/2013-10-04-custdata", "mandatory": true},
    {"url": "s3://mybucket-beta/2013-10-05-custdata", "mandatory": true}
  ]
}
```

The optional `mandatory` flag specifies whether `COPY` should return an error if the file is not found. The default of `mandatory` is `false`. Regardless of any mandatory settings, `COPY` will terminate if no files are found.

The following example runs the COPY command with the manifest in the previous example, which is named `cust.manifest`.

```
copy customer
from 's3://mybucket/cust.manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

Using a manifest created by UNLOAD

A manifest created by an [UNLOAD \(p. 991\)](#) operation using the `MANIFEST` parameter might have keys that are not required for the COPY operation. For example, the following UNLOAD manifest includes a meta key that is required for an Amazon Redshift Spectrum external table and for loading data files in an ORC or Parquet file format. The meta key contains a `content_length` key with a value that is the actual size of the file in bytes. The COPY operation requires only the `url` key and an optional `mandatory` key.

```
{
  "entries": [
    {"url": "s3://mybucket/unload/manifest_0000_part_00", "meta": { "content_length": 5956875 }},
    {"url": "s3://mybucket/unload/unload/manifest_0001_part_00", "meta": { "content_length": 5997091 }}
  ]
}
```

For more information about manifest files, see [Example: COPY from Amazon S3 using a manifest \(p. 727\)](#).

Loading compressed data files from Amazon S3

To load data files that are compressed using gzip, lzop, or bzip2, include the corresponding option: GZIP, LZOP, or BZIP2.

COPY does not support files compressed using the lzop `--filter` option.

For example, the following command loads from files that were compressing using lzop.

```
copy customer from 's3://mybucket/customer.lzo'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' lzop;
```

Loading fixed-width data from Amazon S3

Fixed-width data files have uniform lengths for each column of data. Each field in a fixed-width data file has exactly the same length and position. For character data (CHAR and VARCHAR) in a fixed-width data file, you must include leading or trailing spaces as placeholders in order to keep the width uniform. For integers, you must use leading zeros as placeholders. A fixed-width data file has no delimiter to separate columns.

To load a fixed-width data file into an existing table, USE the `FIXEDWIDTH` parameter in the COPY command. Your table specifications must match the value of `fixedwidth_spec` in order for the data to load correctly.

To load fixed-width data from a file to a table, issue the following command:

```
copy table_name from 's3://mybucket/prefix'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
```

```
fixedwidth 'fixedwidth_spec';
```

The *fixedwidth_spec* parameter is a string that contains an identifier for each column and the width of each column, separated by a colon. The **column:width** pairs are delimited by commas. The identifier can be anything that you choose: numbers, letters, or a combination of the two. The identifier has no relation to the table itself, so the specification must contain the columns in the same order as the table.

The following two examples show the same specification, with the first using numeric identifiers and the second using string identifiers:

```
'0:3,1:25,2:12,3:2,4:6'
```

```
'venueid:3,venuename:25,venuecity:12,venuestate:2,venueseats:6'
```

The following example shows fixed-width sample data that could be loaded into the VENUE table using the preceding specifications:

1	Toyota Park	Bridgeview	IL0
2	Columbus Crew Stadium	Columbus	OH0
3	RFK Stadium	Washington	DC0
4	CommunityAmerica Ballpark	Kansas City	KS0
5	Gillette Stadium	Foxborough	MA68756

The following COPY command loads this data set into the VENUE table:

```
copy venue
from 's3://mybucket/data/venue_fw.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
fixedwidth 'venueid:3,venuename:25,venuecity:12,venuestate:2,venueseats:6';
```

Loading multibyte data from Amazon S3

If your data includes non-ASCII multibyte characters (such as Chinese or Cyrillic characters), you must load the data to VARCHAR columns. The VARCHAR data type supports four-byte UTF-8 characters, but the CHAR data type only accepts single-byte ASCII characters. You cannot load five-byte or longer characters into Amazon Redshift tables. For more information about CHAR and VARCHAR, see [Data types \(p. 574\)](#).

To check which encoding an input file uses, use the Linux `file` command:

```
$ file ordersdata.txt
ordersdata.txt: ASCII English text
$ file uni_ordersdata.dat
uni_ordersdata.dat: UTF-8 Unicode text
```

Loading encrypted data files from Amazon S3

You can use the COPY command to load data files that were uploaded to Amazon S3 using server-side encryption, client-side encryption, or both.

The COPY command supports the following types of Amazon S3 encryption:

- Server-side encryption with Amazon S3-managed keys (SSE-S3)
- Server-side encryption with AWS KMS keys (SSE-KMS)
- Client-side encryption using a client-side symmetric root key

The COPY command doesn't support the following types of Amazon S3 encryption:

- Server-side encryption with customer-provided keys (SSE-C)
- Client-side encryption using an AWS KMS key
- Client-side encryption using a customer-provided asymmetric root key

For more information about Amazon S3 encryption, see [Protecting Data Using Server-Side Encryption](#) and [Protecting Data Using Client-Side Encryption](#) in the Amazon Simple Storage Service User Guide.

The [UNLOAD \(p. 991\)](#) command automatically encrypts files using SSE-S3. You can also unload using SSE-KMS or client-side encryption with a customer managed symmetric key. For more information, see [Unloading encrypted data files \(p. 146\)](#)

The COPY command automatically recognizes and loads files encrypted using SSE-S3 and SSE-KMS. You can load files encrypted using a client-side symmetric root key by specifying the ENCRYPTED option and providing the key value. For more information, see [Uploading encrypted data to Amazon S3 \(p. 68\)](#).

To load client-side encrypted data files, provide the root key value using the MASTER_SYMMETRIC_KEY parameter and include the ENCRYPTED option.

```
copy customer from 's3://mybucket/encrypted/customer'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
encrypted
delimiter '|';
;
```

To load encrypted data files that are gzip, lzop, or bzip2 compressed, include the GZIP, LZOP, or BZIP2 option along with the root key value and the ENCRYPTED option.

```
copy customer from 's3://mybucket/encrypted/customer'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
encrypted
delimiter '|'
gzip;
;
```

Loading data from Amazon EMR

You can use the COPY command to load data in parallel from an Amazon EMR cluster configured to write text files to the cluster's Hadoop Distributed File System (HDFS) as fixed-width files, character-delimited files, CSV files, or JSON-formatted files.

Process for loading data from Amazon EMR

This section walks you through the process of loading data from an Amazon EMR cluster. The following sections provide the details that you must accomplish each step.

- [Step 1: Configure IAM permissions \(p. 75\)](#)

The users that create the Amazon EMR cluster and run the Amazon Redshift COPY command must have the necessary permissions.

- [Step 2: Create an Amazon EMR cluster \(p. 75\)](#)

Configure the cluster to output text files to the Hadoop Distributed File System (HDFS). You will need the Amazon EMR cluster ID and the cluster's main public DNS (the endpoint for the Amazon EC2 instance that hosts the cluster).

- **Step 3: Retrieve the Amazon Redshift cluster public key and cluster node IP addresses (p. 76)**

The public key enables the Amazon Redshift cluster nodes to establish SSH connections to the hosts. You will use the IP address for each cluster node to configure the host security groups to permit access from your Amazon Redshift cluster using these IP addresses.

- **Step 4: Add the Amazon Redshift cluster public key to each Amazon EC2 host's authorized keys file (p. 77)**

You add the Amazon Redshift cluster public key to the host's authorized keys file so that the host will recognize the Amazon Redshift cluster and accept the SSH connection.

- **Step 5: Configure the hosts to accept all of the Amazon Redshift cluster's IP addresses (p. 78)**

Modify the Amazon EMR instance's security groups to add input rules to accept the Amazon Redshift IP addresses.

- **Step 6: Run the COPY command to load the data (p. 78)**

From an Amazon Redshift database, run the COPY command to load the data into an Amazon Redshift table.

Step 1: Configure IAM permissions

The users that create the Amazon EMR cluster and run the Amazon Redshift COPY command must have the necessary permissions.

To configure IAM permissions

1. Add the following permissions for the user that will create the Amazon EMR cluster.

```
ec2:DescribeSecurityGroups  
ec2:RevokeSecurityGroupIngress  
ec2:AuthorizeSecurityGroupIngress  
redshift:DescribeClusters
```

2. Add the following permission for the IAM role or user that will run the COPY command.

```
elasticmapreduce>ListInstances
```

3. Add the following permission to the Amazon EMR cluster's IAM role.

```
redshift:DescribeClusters
```

Step 2: Create an Amazon EMR cluster

The COPY command loads data from files on the Amazon EMR Hadoop Distributed File System (HDFS). When you create the Amazon EMR cluster, configure the cluster to output data files to the cluster's HDFS.

To create an Amazon EMR cluster

1. Create an Amazon EMR cluster in the same AWS Region as the Amazon Redshift cluster.

If the Amazon Redshift cluster is in a VPC, the Amazon EMR cluster must be in the same VPC group.

If the Amazon Redshift cluster uses EC2-Classic mode (that is, it is not in a VPC), the Amazon EMR cluster must also use EC2-Classic mode. For more information, see [Managing Clusters in Virtual Private Cloud \(VPC\)](#) in the *Amazon Redshift Management Guide*.

2. Configure the cluster to output data files to the cluster's HDFS. The HDFS file names must not include asterisks (*) or question marks (?).

Important

The file names must not include asterisks (*) or question marks (?).

3. Specify **No** for the **Auto-terminate** option in the Amazon EMR cluster configuration so that the cluster remains available while the COPY command runs.

Important

If any of the data files are changed or deleted before the COPY completes, you might have unexpected results, or the COPY operation might fail.

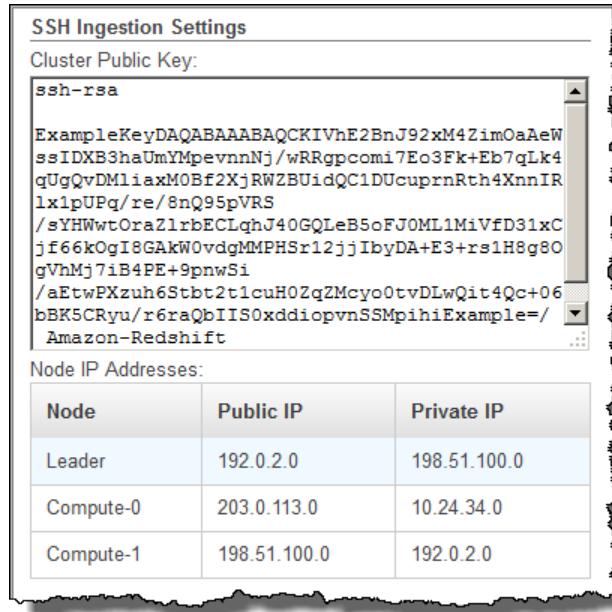
4. Note the cluster ID and the main public DNS (the endpoint for the Amazon EC2 instance that hosts the cluster). You will use that information in later steps.

Step 3: Retrieve the Amazon Redshift cluster public key and cluster node IP addresses

To retrieve the Amazon Redshift cluster public key and cluster node IP addresses for your cluster using the console

1. Access the Amazon Redshift Management Console.
2. Choose the **Clusters** link in the navigation pane.
3. Select your cluster from the list.
4. Locate the **SSH Ingestion Settings** group.

Note the **Cluster Public Key** and **Node IP addresses**. You will use them in later steps.



You will use the private IP addresses in Step 3 to configure the Amazon EC2 host to accept the connection from Amazon Redshift.

To retrieve the cluster public key and cluster node IP addresses for your cluster using the Amazon Redshift CLI, run the describe-clusters command. For example:

```
aws redshift describe-clusters --cluster-identifier <cluster-identifier>
```

The response will include a ClusterPublicKey value and the list of private and public IP addresses, similar to the following:

```
{  
    "Clusters": [  
        {  
            "VpcSecurityGroups": [],  
            "ClusterStatus": "available",  
            "ClusterNodes": [  
                {  
                    "PrivateIPAddress": "10.nnn.nnn.nnn",  
                    "NodeRole": "LEADER",  
                    "PublicIPAddress": "10.nnn.nnn.nnn"  
                },  
                {  
                    "PrivateIPAddress": "10.nnn.nnn.nnn",  
                    "NodeRole": "COMPUTE-0",  
                    "PublicIPAddress": "10.nnn.nnn.nnn"  
                },  
                {  
                    "PrivateIPAddress": "10.nnn.nnn.nnn",  
                    "NodeRole": "COMPUTE-1",  
                    "PublicIPAddress": "10.nnn.nnn.nnn"  
                }  
            ],  
            "AutomatedSnapshotRetentionPeriod": 1,  
            "PreferredMaintenanceWindow": "wed:05:30-wed:06:00",  
            "AvailabilityZone": "us-east-1a",  
            "NodeType": "ds2.xlarge",  
            "ClusterPublicKey": "ssh-rsa AAAABexamplepublickey...Y3TA1 Amazon-Redshift",  
            ...  
            ...  
        }  
    ]  
}
```

To retrieve the cluster public key and cluster node IP addresses for your cluster using the Amazon Redshift API, use the `DescribeClusters` action. For more information, see [describe-clusters](#) in the *Amazon Redshift CLI Guide* or [DescribeClusters](#) in the Amazon Redshift API Guide.

Step 4: Add the Amazon Redshift cluster public key to each Amazon EC2 host's authorized keys file

You add the cluster public key to each host's authorized keys file for all of the Amazon EMR cluster nodes so that the hosts will recognize Amazon Redshift and accept the SSH connection.

To add the Amazon Redshift cluster public key to the host's authorized keys file

1. Access the host using an SSH connection.

For information about connecting to an instance using SSH, see [Connect to Your Instance](#) in the *Amazon EC2 User Guide*.

2. Copy the Amazon Redshift public key from the console or from the CLI response text.
3. Copy and paste the contents of the public key into the `/home/<ssh_username>/.ssh/authorized_keys` file on the host. Include the complete string, including the prefix `"ssh-rsa "` and suffix `"Amazon-Redshift"`. For example:

```
ssh-rsa AAAACTP3isxgGzVWoIWpbVvRC0zYdVifMrh... uA70BnMHCaMiRdmvsD0edZD0edZ Amazon-Redshift
```

Step 5: Configure the hosts to accept all of the Amazon Redshift cluster's IP addresses

To allow inbound traffic to the host instances, edit the security group and add one Inbound rule for each Amazon Redshift cluster node. For **Type**, select SSH with TCP protocol on Port 22. For **Source**, enter the Amazon Redshift cluster node private IP addresses you retrieved in [Step 3: Retrieve the Amazon Redshift cluster public key and cluster node IP addresses \(p. 76\)](#). For information about adding rules to an Amazon EC2 security group, see [Authorizing Inbound Traffic for Your Instances](#) in the *Amazon EC2 User Guide*.

Step 6: Run the COPY command to load the data

Run a [COPY \(p. 677\)](#) command to connect to the Amazon EMR cluster and load the data into an Amazon Redshift table. The Amazon EMR cluster must continue running until the COPY command completes. For example, do not configure the cluster to auto-terminate.

Important

If any of the data files are changed or deleted before the COPY completes, you might have unexpected results, or the COPY operation might fail.

In the COPY command, specify the Amazon EMR cluster ID and the HDFS file path and file name.

```
copy sales
from 'emr://myemrclusterid/myoutput/part*' credentials
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

You can use the wildcard characters asterisk (*) and question mark (?) as part of the file name argument. For example, part* loads the files part-0000, part-0001, and so on. If you specify only a folder name, COPY attempts to load all files in the folder.

Important

If you use wildcard characters or use only the folder name, verify that no unwanted files will be loaded or the COPY command will fail. For example, some processes might write a log file to the output folder.

Loading data from remote hosts

You can use the COPY command to load data in parallel from one or more remote hosts, such as Amazon EC2 instances or other computers. COPY connects to the remote hosts using SSH and runs commands on the remote hosts to generate text output.

The remote host can be an Amazon EC2 Linux instance or another Unix or Linux computer configured to accept SSH connections. This guide assumes your remote host is an Amazon EC2 instance. Where the procedure is different for another computer, the guide will point out the difference.

Amazon Redshift can connect to multiple hosts, and can open multiple SSH connections to each host. Amazon Redshift sends a unique command through each connection to generate text output to the host's standard output, which Amazon Redshift then reads as it would a text file.

Before you begin

Before you begin, you should have the following in place:

- One or more host machines, such as Amazon EC2 instances, that you can connect to using SSH.
- Data sources on the hosts.

You will provide commands that the Amazon Redshift cluster will run on the hosts to generate the text output. After the cluster connects to a host, the COPY command runs the commands, reads the text from the hosts' standard output, and loads the data in parallel into an Amazon Redshift table. The text output must be in a form that the COPY command can ingest. For more information, see [Preparing your input data \(p. 65\)](#)

- Access to the hosts from your computer.

For an Amazon EC2 instance, you will use an SSH connection to access the host. You must access the host to add the Amazon Redshift cluster's public key to the host's authorized keys file.

- A running Amazon Redshift cluster.

For information about how to launch a cluster, see [Amazon Redshift Getting Started Guide](#).

Loading data process

This section walks you through the process of loading data from remote hosts. The following sections provide the details that you must accomplish in each step.

- [**Step 1: Retrieve the cluster public key and cluster node IP addresses \(p. 80\)**](#)

The public key enables the Amazon Redshift cluster nodes to establish SSH connections to the remote hosts. You will use the IP address for each cluster node to configure the host security groups or firewall to permit access from your Amazon Redshift cluster using these IP addresses.

- [**Step 2: Add the Amazon Redshift cluster public key to the host's authorized keys file \(p. 81\)**](#)

You add the Amazon Redshift cluster public key to the host's authorized keys file so that the host will recognize the Amazon Redshift cluster and accept the SSH connection.

- [**Step 3: Configure the host to accept all of the Amazon Redshift cluster's IP addresses \(p. 81\)**](#)

For Amazon EC2, modify the instance's security groups to add input rules to accept the Amazon Redshift IP addresses. For other hosts, modify the firewall so that your Amazon Redshift nodes are able to establish SSH connections to the remote host.

- [**Step 4: Get the public key for the host \(p. 82\)**](#)

You can optionally specify that Amazon Redshift should use the public key to identify the host. You must locate the public key and copy the text into your manifest file.

- [**Step 5: Create a manifest file \(p. 82\)**](#)

The manifest is a JSON-formatted text file with the details Amazon Redshift needs to connect to the hosts and fetch the data.

- [**Step 6: Upload the manifest file to an Amazon S3 bucket \(p. 84\)**](#)

Amazon Redshift reads the manifest and uses that information to connect to the remote host. If the Amazon S3 bucket does not reside in the same Region as your Amazon Redshift cluster, you must use the [REGION \(p. 686\)](#) option to specify the Region in which the data is located.

- [**Step 7: Run the COPY command to load the data \(p. 84\)**](#)

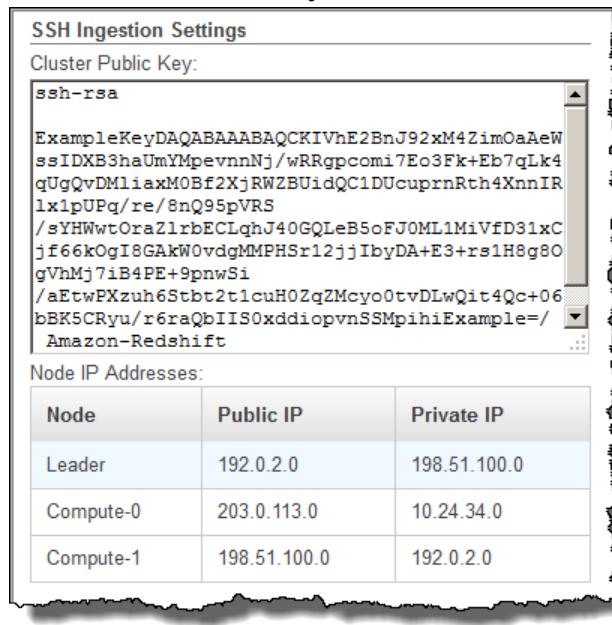
From an Amazon Redshift database, run the COPY command to load the data into an Amazon Redshift table.

Step 1: Retrieve the cluster public key and cluster node IP addresses

To retrieve the cluster public key and cluster node IP addresses for your cluster using the console

1. Access the Amazon Redshift Management Console.
2. Choose the **Clusters** link in the navigation pane.
3. Select your cluster from the list.
4. Locate the **SSH Ingestion Settings** group.

Note the **Cluster Public Key** and **Node IP addresses**. You will use them in later steps.



Node	Public IP	Private IP
Leader	192.0.2.0	198.51.100.0
Compute-0	203.0.113.0	10.24.34.0
Compute-1	198.51.100.0	192.0.2.0

You will use the IP addresses in Step 3 to configure the host to accept the connection from Amazon Redshift. Depending on what type of host you connect to and whether it is in a VPC, you will use either the public IP addresses or the private IP addresses.

To retrieve the cluster public key and cluster node IP addresses for your cluster using the Amazon Redshift CLI, run the `describe-clusters` command.

For example:

```
aws redshift describe-clusters --cluster-identifier <cluster-identifier>
```

The response will include the `ClusterPublicKey` and the list of private and public IP addresses, similar to the following:

```
{  
    "Clusters": [  
        {  
            "VpcSecurityGroups": [],  
            "ClusterStatus": "available",  
            "ClusterNodes": [  
                {  
                    "PublicIpAddress": "192.0.2.0",  
                    "PrivateIpAddress": "198.51.100.0",  
                    "NodeType": "Leader",  
                    "Port": 5439  
                },  
                {  
                    "PublicIpAddress": "203.0.113.0",  
                    "PrivateIpAddress": "10.24.34.0",  
                    "NodeType": "Compute-0",  
                    "Port": 5439  
                },  
                {  
                    "PublicIpAddress": "198.51.100.0",  
                    "PrivateIpAddress": "192.0.2.0",  
                    "NodeType": "Compute-1",  
                    "Port": 5439  
                }  
            ]  
        }  
    ]  
}
```

```
{
    "PrivateIPAddress": "10.nnn.nnn.nnn",
    "NodeRole": "LEADER",
    "PublicIPAddress": "10.nnn.nnn.nnn"
},
{
    "PrivateIPAddress": "10.nnn.nnn.nnn",
    "NodeRole": "COMPUTE-0",
    "PublicIPAddress": "10.nnn.nnn.nnn"
},
{
    "PrivateIPAddress": "10.nnn.nnn.nnn",
    "NodeRole": "COMPUTE-1",
    "PublicIPAddress": "10.nnn.nnn.nnn"
},
],
"AutomatedSnapshotRetentionPeriod": 1,
"PreferredMaintenanceWindow": "wed:05:30-wed:06:00",
"AvailabilityZone": "us-east-1a",
"NodeType": "ds2.xlarge",
"ClusterPublicKey": "ssh-rsa AAAABexamplepublickey...Y3TA1 Amazon-Redshift",
...
}
```

To retrieve the cluster public key and cluster node IP addresses for your cluster using the Amazon Redshift API, use the `DescribeClusters` action. For more information, see [describe-clusters](#) in the *Amazon Redshift CLI Guide* or [DescribeClusters](#) in the Amazon Redshift API Guide.

Step 2: Add the Amazon Redshift cluster public key to the host's authorized keys file

You add the cluster public key to each host's authorized keys file so that the host will recognize Amazon Redshift and accept the SSH connection.

To add the Amazon Redshift cluster public key to the host's authorized keys file

1. Access the host using an SSH connection.

For information about connecting to an instance using SSH, see [Connect to Your Instance](#) in the *Amazon EC2 User Guide*.

2. Copy the Amazon Redshift public key from the console or from the CLI response text.
3. Copy and paste the contents of the public key into the `/home/<ssh_username>/.``ssh/authorized_keys` file on the remote host. The `<ssh_username>` must match the value for the "username" field in the manifest file. Include the complete string, including the prefix "ssh-rsa" and suffix "Amazon-Redshift". For example:

```
ssh-rsa AAAACTP3isxgGzVWoIWpbVvRC0zYdVifMh... uA70BnMHCaMiRdmvsD0edZD0edZ Amazon-Redshift
```

Step 3: Configure the host to accept all of the Amazon Redshift cluster's IP addresses

If you are working with an Amazon EC2 instance or an Amazon EMR cluster, add Inbound rules to the host's security group to allow traffic from each Amazon Redshift cluster node. For **Type**, select SSH with TCP protocol on Port 22. For **Source**, enter the Amazon Redshift cluster node IP addresses you retrieved

in [Step 1: Retrieve the cluster public key and cluster node IP addresses \(p. 80\)](#). For information about adding rules to an Amazon EC2 security group, see [Authorizing Inbound Traffic for Your Instances](#) in the [Amazon EC2 User Guide](#).

Use the private IP addresses when:

- You have an Amazon Redshift cluster that is not in a Virtual Private Cloud (VPC), and an Amazon EC2 - Classic instance, both of which are in the same AWS Region.
- You have an Amazon Redshift cluster that is in a VPC, and an Amazon EC2 -VPC instance, both of which are in the same AWS Region and in the same VPC.

Otherwise, use the public IP addresses.

For more information about using Amazon Redshift in a VPC, see [Managing Clusters in Virtual Private Cloud \(VPC\)](#) in the [Amazon Redshift Management Guide](#).

Step 4: Get the public key for the host

You can optionally provide the host's public key in the manifest file so that Amazon Redshift can identify the host. The COPY command does not require the host public key but, for security reasons, we strongly recommend using a public key to help prevent 'man-in-the-middle' attacks.

You can find the host's public key in the following location, where <ssh_host_rsa_key_name> is the unique name for the host's public key:

```
: /etc/ssh/<ssh_host_rsa_key_name>.pub
```

Note

Amazon Redshift only supports RSA keys. We do not support DSA keys.

When you create your manifest file in Step 5, you will paste the text of the public key into the "Public Key" field in the manifest file entry.

Step 5: Create a manifest file

The COPY command can connect to multiple hosts using SSH, and can create multiple SSH connections to each host. COPY runs a command through each host connection, and then loads the output from the commands in parallel into the table. The manifest file is a text file in JSON format that Amazon Redshift uses to connect to the host. The manifest file specifies the SSH host endpoints and the commands that are run on the hosts to return data to Amazon Redshift. Optionally, you can include the host public key, the login user name, and a mandatory flag for each entry.

Create the manifest file on your local computer. In a later step, you upload the file to Amazon S3.

The manifest file is in the following format:

```
{  
  "entries": [  
    {"endpoint": "<ssh_endpoint_or_IP>",  
     "command": "<remote_command>",  
     "mandatory": true,  
     "publickey": "<public_key>",  
     "username": "<host_user_name>"},  
    {"endpoint": "<ssh_endpoint_or_IP>",  
     "command": "<remote_command>",  
     "mandatory": true,  
     "publickey": "<public_key>",  
     "username": "<host_user_name>"}]
```

```

        "username": "host_user_name"
    ]
}

```

The manifest file contains one "entries" construct for each SSH connection. Each entry represents a single SSH connection. You can have multiple connections to a single host or multiple connections to multiple hosts. The double quotation marks are required as shown, both for the field names and the values. The only value that does not need double quotation marks is the Boolean value **true** or **false** for the mandatory field.

The following describes the fields in the manifest file.

endpoint

The URL address or IP address of the host. For example, "ec2-111-222-333.compute-1.amazonaws.com" or "22.33.44.56"

command

The command that will be run by the host to generate text or binary (gzip, lzop, or bzip2) output. The command can be any command that the user "*host_user_name*" has permission to run. The command can be as simple as printing a file, or it could query a database or launch a script. The output (text file, gzip binary file, lzop binary file, or bzip2 binary file) must be in a form the Amazon Redshift COPY command can ingest. For more information, see [Preparing your input data \(p. 65\)](#).

publickey

(Optional) The public key of the host. If provided, Amazon Redshift will use the public key to identify the host. If the public key is not provided, Amazon Redshift will not attempt host identification. For example, if the remote host's public key is: ssh-rsa AbcCbaxxx...xxxDHKJ root@amazon.com, enter the following text in the public key field: AbcCbaxxx...xxxDHKJ.

mandatory

(Optional) Indicates whether the COPY command should fail if the connection fails. The default is **false**. If Amazon Redshift does not successfully make at least one connection, the COPY command fails.

username

(Optional) The username that will be used to log on to the host system and run the remote command. The user login name must be the same as the login that was used to add the public key to the host's authorized keys file in Step 2. The default username is "redshift".

The following example shows a completed manifest to open four connections to the same host and run a different command through each connection:

```
{
  "entries": [
    {"endpoint": "ec2-184-72-204-112.compute-1.amazonaws.com",
     "command": "cat loaddata1.txt",
     "mandatory": true,
     "publickey": "ec2publickeyportionoftheec2keypair",
     "username": "ec2-user"},
    {"endpoint": "ec2-184-72-204-112.compute-1.amazonaws.com",
     "command": "cat loaddata2.txt",
     "mandatory": true,
     "publickey": "ec2publickeyportionoftheec2keypair",
     "username": "ec2-user"},
    {"endpoint": "ec2-184-72-204-112.compute-1.amazonaws.com",
     "command": "cat loaddata3.txt",
     "mandatory": true,
     "publickey": "ec2publickeyportionoftheec2keypair",
     "username": "ec2-user"}
  ]
}
```

```
        "publickey": "ec2publickeyportionoftheec2keypair",
        "username": "ec2-user"},  
    {"endpoint": "ec2-184-72-204-112.compute-1.amazonaws.com",
     "command": "cat loaddata4.txt",
     "mandatory": true,
     "publickey": "ec2publickeyportionoftheec2keypair",
     "username": "ec2-user"}  
]  
}
```

Step 6: Upload the manifest file to an Amazon S3 bucket

Upload the manifest file to an Amazon S3 bucket. If the Amazon S3 bucket does not reside in the same AWS Region as your Amazon Redshift cluster, you must use the [REGION \(p. 686\)](#) option to specify the AWS Region in which the manifest is located. For information about creating an Amazon S3 bucket and uploading a file, see [Amazon Simple Storage Service User Guide](#).

Step 7: Run the COPY command to load the data

Run a [COPY \(p. 677\)](#) command to connect to the host and load the data into an Amazon Redshift table. In the COPY command, specify the explicit Amazon S3 object path for the manifest file and include the SSH option. For example,

```
copy sales
from 's3://mybucket/ssh_manifest' credentials
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|'
ssh;
```

Note

If you use automatic compression, the COPY command performs two data reads, which means it runs the remote command twice. The first read is to provide a sample for compression analysis, then the second read actually loads the data. If running the remote command twice might cause a problem because of potential side effects, you should turn off automatic compression. To turn off automatic compression, run the COPY command with the COMPUPDATE option set to OFF.

For more information, see [Loading tables with automatic compression \(p. 87\)](#).

Loading data from an Amazon DynamoDB table

You can use the COPY command to load a table with data from a single Amazon DynamoDB table.

Important

The Amazon DynamoDB table that provides the data must be created in the same AWS Region as your cluster unless you use the [REGION \(p. 686\)](#) option to specify the AWS Region in which the Amazon DynamoDB table is located.

The COPY command uses the Amazon Redshift massively parallel processing (MPP) architecture to read and load data in parallel from an Amazon DynamoDB table. You can take maximum advantage of parallel processing by setting distribution styles on your Amazon Redshift tables. For more information, see [Working with data distribution styles \(p. 49\)](#).

Important

When the COPY command reads data from the Amazon DynamoDB table, the resulting data transfer is part of that table's provisioned throughput.

To avoid consuming excessive amounts of provisioned read throughput, we recommend that you not load data from Amazon DynamoDB tables that are in production environments. If you do load data from production tables, we recommend that you set the READRATIO option much lower than the average

percentage of unused provisioned throughput. A low READRATIO setting will help minimize throttling issues. To use the entire provisioned throughput of an Amazon DynamoDB table, set READRATIO to 100.

The COPY command matches attribute names in the items retrieved from the DynamoDB table to column names in an existing Amazon Redshift table by using the following rules:

- Amazon Redshift table columns are case-insensitively matched to Amazon DynamoDB item attributes. If an item in the DynamoDB table contains multiple attributes that differ only in case, such as Price and PRICE, the COPY command will fail.
- Amazon Redshift table columns that do not match an attribute in the Amazon DynamoDB table are loaded as either NULL or empty, depending on the value specified with the EMPTYASNULL option in the [COPY \(p. 677\)](#) command.
- Amazon DynamoDB attributes that do not match a column in the Amazon Redshift table are discarded. Attributes are read before they are matched, and so even discarded attributes consume part of that table's provisioned throughput.
- Only Amazon DynamoDB attributes with scalar STRING and NUMBER data types are supported. The Amazon DynamoDB BINARY and SET data types are not supported. If a COPY command tries to load an attribute with an unsupported data type, the command will fail. If the attribute does not match an Amazon Redshift table column, COPY does not attempt to load it, and it does not raise an error.

The COPY command uses the following syntax to load data from an Amazon DynamoDB table:

```
copy <redshift_tablename> from 'dynamodb://<dynamodb_table_name>'  
authorization  
readratio '<integer>';
```

The values for *authorization* are the AWS credentials needed to access the Amazon DynamoDB table. If these credentials correspond to a user, that user must have permission to SCAN and DESCRIBE the Amazon DynamoDB table that is being loaded.

The values for *authorization* provide the AWS authorization your cluster needs to access the Amazon DynamoDB table. The permission must include SCAN and DESCRIBE for the Amazon DynamoDB table that is being loaded. For more information about required permissions, see [IAM permissions for COPY, UNLOAD, and CREATE LIBRARY \(p. 718\)](#). The preferred method for authentication is to specify the IAM_ROLE parameter and provide the Amazon Resource Name (ARN) for an IAM role with the necessary permissions. For more information, see [Role-based access control \(p. 715\)](#).

To authenticate using the IAM_ROLE parameter, `<aws-account-id>` and `<role-name>` as shown in the following syntax.

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

The following example shows authentication using an IAM role.

```
copy favoritemovies  
from 'dynamodb://ProductCatalog'  
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

For more information about other authorization options, see [Authorization parameters \(p. 693\)](#)

If you want to validate your data without actually loading the table, use the NOLOAD option with the [COPY \(p. 677\)](#) command.

The following example loads the FAVORITEMOVIES table with data from the DynamoDB table my-favorite-movies-table. The read activity can consume up to 50% of the provisioned throughput.

```
copy favoritemovies from 'dynamodb://my-favorite-movies-table'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
readratio 50;
```

To maximize throughput, the COPY command loads data from an Amazon DynamoDB table in parallel across the compute nodes in the cluster.

Provisioned throughput with automatic compression

By default, the COPY command applies automatic compression whenever you specify an empty target table with no compression encoding. The automatic compression analysis initially samples a large number of rows from the Amazon DynamoDB table. The sample size is based on the value of the COMPROWS parameter. The default is 100,000 rows per slice.

After sampling, the sample rows are discarded and the entire table is loaded. As a result, many rows are read twice. For more information about how automatic compression works, see [Loading tables with automatic compression \(p. 87\)](#).

Important

When the COPY command reads data from the Amazon DynamoDB table, including the rows used for sampling, the resulting data transfer is part of that table's provisioned throughput.

Loading multibyte data from Amazon DynamoDB

If your data includes non-ASCII multibyte characters (such as Chinese or Cyrillic characters), you must load the data to VARCHAR columns. The VARCHAR data type supports four-byte UTF-8 characters, but the CHAR data type only accepts single-byte ASCII characters. You cannot load five-byte or longer characters into Amazon Redshift tables. For more information about CHAR and VARCHAR, see [Data types \(p. 574\)](#).

Verifying that the data loaded correctly

After the load operation is complete, query the [STL_LOAD_COMMITS \(p. 1523\)](#) system table to verify that the expected files were loaded. Run the COPY command and load verification within the same transaction so that if there is problem with the load you can roll back the entire transaction.

The following query returns entries for loading the tables in the TICKIT database:

```
select query, trim(filename) as filename, curtime, status
from stl_load_commits
where filename like '%tickit%' order by query;
```

query	filename	curtime	status
22475	tickit/allusers_pipe.txt	2013-02-08 20:58:23.274186	1
22478	tickit/venue_pipe.txt	2013-02-08 20:58:25.070604	1
22480	tickit/category_pipe.txt	2013-02-08 20:58:27.333472	1
22482	tickit/date2008_pipe.txt	2013-02-08 20:58:28.608305	1
22485	tickit/allevents_pipe.txt	2013-02-08 20:58:29.99489	1
22487	tickit/listings_pipe.txt	2013-02-08 20:58:37.632939	1
22489	tickit/sales_tab.txt	2013-02-08 20:58:37.632939	1

(6 rows)

Validating input data

To validate the data in the Amazon S3 input files or Amazon DynamoDB table before you actually load the data, use the NOLOAD option with the [COPY \(p. 677\)](#) command. Use NOLOAD with the same COPY

commands and options you would use to load the data. NOLOAD checks the integrity of all of the data without loading it into the database. The NOLOAD option displays any errors that occur if you attempt to load the data.

For example, if you specified the incorrect Amazon S3 path for the input file, Amazon Redshift would display the following error.

```
ERROR: No such file or directory
DETAIL:
-----
Amazon Redshift error: The specified key does not exist
code: 2
context: S3 key being read :
location: step_scan.cpp:1883
process: xenmaster [pid=22199]
-----
```

To troubleshoot error messages, see the [Load error reference \(p. 93\)](#).

For an example using the NOLOAD option, see [COPY command with the NOLOAD option \(p. 745\)](#).

Loading tables with automatic compression

Topics

- [How automatic compression works \(p. 87\)](#)
- [Automatic compression example \(p. 88\)](#)

You can apply compression encodings to columns in tables manually, based on your own evaluation of the data. Or you can use the COPY command with COMPUPDATE set to ON to analyze and apply compression automatically based on sample data.

You can use automatic compression when you create and load a brand new table. The COPY command performs a compression analysis. You can also perform a compression analysis without loading data or changing the compression on a table by running the [ANALYZE COMPRESSION \(p. 665\)](#) command on an already populated table. For example, you can run ANALYZE COMPRESSION when you want to analyze compression on a table for future use, while preserving the existing data definition language (DDL) statements.

Automatic compression balances overall performance when choosing compression encodings. Range-restricted scans might perform poorly if sort key columns are compressed much more highly than other columns in the same query. As a result, automatic compression skips the data analyzing phase on the sort key columns and keeps the user-defined encoding types.

Automatic compression chooses RAW encoding if you haven't explicitly defined a type of encoding. ANALYZE COMPRESSION behaves the same. For optimal query performance, consider using RAW for sort keys.

How automatic compression works

When the COMPUPDATE parameter is ON, the COPY command applies automatic compression whenever you run the COPY command with an empty target table and all of the table columns either have RAW encoding or no encoding.

To apply automatic compression to an empty table, regardless of its current compression encodings, run the COPY command with the COMPUPDATE option set to ON. To turn off automatic compression, run the COPY command with the COMPUPDATE option set to OFF.

You cannot apply automatic compression to a table that already contains data.

Note

Automatic compression analysis requires enough rows in the load data (at least 100,000 rows per slice) to generate a meaningful sample.

Automatic compression performs these operations in the background as part of the load transaction:

1. An initial sample of rows is loaded from the input file. Sample size is based on the value of the COMPROWS parameter. The default is 100,000.
2. Compression options are chosen for each column.
3. The sample rows are removed from the table.
4. The table is recreated with the chosen compression encodings.
5. The entire input file is loaded and compressed using the new encodings.

After you run the COPY command, the table is fully loaded, compressed, and ready for use. If you load more data later, appended rows are compressed according to the existing encoding.

If you only want to perform a compression analysis, run ANALYZE COMPRESSION, which is more efficient than running a full COPY. Then you can evaluate the results to decide whether to use automatic compression or recreate the table manually.

Automatic compression is supported only for the COPY command. Alternatively, you can manually apply compression encoding when you create the table. For information about manual compression encoding, see [Working with column compression \(p. 38\)](#).

Automatic compression example

In this example, assume that the TICKIT database contains a copy of the LISTING table called BIGLIST, and you want to apply automatic compression to this table when it is loaded with approximately 3 million rows.

To load and automatically compress the table

1. Make sure that the table is empty. You can apply automatic compression only to an empty table:

```
truncate biglist;
```

2. Load the table with a single COPY command. Although the table is empty, some earlier encoding might have been specified. To facilitate that Amazon Redshift performs a compression analysis, set the COMPUPDATE parameter to ON.

```
copy biglist from 's3://mybucket/biglist.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' COMPUPDATE ON;
```

Because no COMPROWS option is specified, the default and recommended sample size of 100,000 rows per slice is used.

3. Look at the new schema for the BIGLIST table in order to review the automatically chosen encoding schemes.

```
select "column", type, encoding
from pg_table_def where tablename = 'biglist';
```

Column	Type	Encoding
--------	------	----------

listid	integer	delta
sellerid	integer	delta32k
eventid	integer	delta32k
dateid	smallint	delta
+numtickets	smallint	delta
priceperticket	numeric(8,2)	delta32k
totalprice	numeric(8,2)	mostly32
listtime	timestamp without time zone	none

- Verify that the expected number of rows were loaded:

```
select count(*) from biglist;
count
-----
3079952
(1 row)
```

When rows are later appended to this table using COPY or INSERT statements, the same compression encodings are applied.

Optimizing storage for narrow tables

If you have a table with very few columns but a very large number of rows, the three hidden metadata identity columns (INSERT_XID, DELETE_XID, ROW_ID) will consume a disproportionate amount of the disk space for the table.

In order to optimize compression of the hidden columns, load the table in a single COPY transaction where possible. If you load the table with multiple separate COPY commands, the INSERT_XID column will not compress well. You must perform a vacuum operation if you use multiple COPY commands, but it will not improve compression of INSERT_XID.

Loading default column values

You can optionally define a column list in your COPY command. If a column in the table is omitted from the column list, COPY will load the column with either the value supplied by the DEFAULT option that was specified in the CREATE TABLE command, or with NULL if the DEFAULT option was not specified.

If COPY attempts to assign NULL to a column that is defined as NOT NULL, the COPY command fails. For information about assigning the DEFAULT option, see [CREATE TABLE \(p. 823\)](#).

When loading from data files on Amazon S3, the columns in the column list must be in the same order as the fields in the data file. If a field in the data file does not have a corresponding column in the column list, the COPY command fails.

When loading from Amazon DynamoDB table, order does not matter. Any fields in the Amazon DynamoDB attributes that do not match a column in the Amazon Redshift table are discarded.

The following restrictions apply when using the COPY command to load DEFAULT values into a table:

- If an [IDENTITY \(p. 826\)](#) column is included in the column list, the EXPLICIT_IDS option must also be specified in the [COPY \(p. 677\)](#) command, or the COPY command will fail. Similarly, if an IDENTITY column is omitted from the column list, and the EXPLICIT_IDS option is specified, the COPY operation will fail.
- Because the evaluated DEFAULT expression for a given column is the same for all loaded rows, a DEFAULT expression that uses a RANDOM() function will assign the same value to all the rows.

- DEFAULT expressions that contain CURRENT_DATE or SYSDATE are set to the timestamp of the current transaction.

For an example, see "Load data from a file with default values" in [COPY examples \(p. 726\)](#).

Troubleshooting data loads

Topics

- [S3ServiceException errors \(p. 90\)](#)
- [System tables for troubleshooting data loads \(p. 91\)](#)
- [Multibyte character load errors \(p. 92\)](#)
- [Load error reference \(p. 93\)](#)

This section provides information about identifying and resolving data loading errors.

S3ServiceException errors

The most common s3ServiceException errors are caused by an improperly formatted or incorrect credentials string, having your cluster and your bucket in different AWS Regions, and insufficient Amazon S3 permissions.

The section provides troubleshooting information for each type of error.

Invalid credentials string

If your credentials string was improperly formatted, you will receive the following error message:

```
ERROR: Invalid credentials. Must be of the format: credentials
'aws_access_key_id=<access-key-id>;aws_secret_access_key=<secret-access-key>
[;token=<temporary-session-token>]'
```

Verify that the credentials string does not contain any spaces or line breaks, and is enclosed in single quotation marks.

Invalid access key ID

If your access key ID does not exist, you will receive the following error message:

```
[Amazon](500310) Invalid operation: S3ServiceException:The AWS Access Key Id you provided
does not exist in our records.
```

This is often a copy and paste error. Verify that the access key ID was entered correctly. Also, if you are using temporary session keys, check that the value for token is set.

Invalid secret access key

If your secret access key is incorrect, you will receive the following error message:

```
[Amazon](500310) Invalid operation: S3ServiceException:The request signature we calculated
does not match the signature you provided.
Check your key and signing method.,Status 403,Error SignatureDoesNotMatch
```

This is often a copy and paste error. Verify that the secret access key was entered correctly and that it is the correct key for the access key ID.

Bucket is in a different Region

The Amazon S3 bucket specified in the COPY command must be in the same AWS Region as the cluster. If your Amazon S3 bucket and your cluster are in different Regions, you will receive an error similar to the following:

```
ERROR: S3ServiceException:The bucket you are attempting to access must be addressed using the specified endpoint.
```

You can create an Amazon S3 bucket in a specific Region either by selecting the Region when you create the bucket by using the Amazon S3 Management Console, or by specifying an endpoint when you create the bucket using the Amazon S3 API or CLI. For more information, see [Uploading files to Amazon S3 \(p. 67\)](#).

For more information about Amazon S3 regions, see [Accessing a Bucket](#) in the *Amazon Simple Storage Service User Guide*.

Alternatively, you can specify the Region using the [REGION \(p. 686\)](#) option with the COPY command.

Access denied

If the user does not have sufficient permissions, you will receive the following error message:

```
ERROR: S3ServiceException:Access Denied,Status 403,Error AccessDenied
```

One possible cause is the user identified by the credentials does not have LIST and GET access to the Amazon S3 bucket. For other causes, see [Troubleshoot Access Denied \(403 Forbidden\) errors in Amazon S3 in the Amazon Simple Storage Service User Guide](#).

For information about managing user access to buckets, see [Identity and access management in Amazon S3 in the Amazon Simple Storage Service User Guide](#).

System tables for troubleshooting data loads

The following Amazon Redshift system tables can be helpful in troubleshooting data load issues:

- Query [STL_LOAD_ERRORS \(p. 1526\)](#) to discover the errors that occurred during specific loads.
- Query [STL_FILE_SCAN \(p. 1516\)](#) to view load times for specific files or to see if a specific file was even read.
- Query [STL_S3CLIENT_ERROR \(p. 1551\)](#) to find details for errors encountered while transferring data from Amazon S3.

To find and diagnose load errors

1. Create a view or define a query that returns details about load errors. The following example joins the STL_LOAD_ERRORS table to the STV_TBL_PERM table to match table IDs with actual table names.

```
create view loadview as
(select distinct tbl, trim(name) as table_name, query, starttime,
trim(filename) as input, line_number, colname, err_code,
trim(err_reason) as reason
from stl_load_errors sl, stv_tbl_perm sp
where sl.tbl = sp.id);
```

2. Set the MAXERRORS option in your COPY command to a large enough value to enable COPY to return useful information about your data. If the COPY encounters errors, an error message directs you to consult the STL_LOAD_ERRORS table for details.

3. Query the LOADVIEW view to see error details. For example:

```
select * from loadview where table_name='venue';
```

tbl	table_name	query	starttime		
100551	venue	20974	2013-01-29 19:05:58.365391		
	input	line_number	colname	err_code	reason
	venue_pipe.txt	1	0	1214	Delimiter not found

4. Fix the problem in the input file or the load script, based on the information that the view returns. Some typical load errors to watch for include:

- Mismatch between data types in table and values in input data fields.
- Mismatch between number of columns in table and number of fields in input data.
- Mismatched quotation marks. Amazon Redshift supports both single and double quotation marks; however, these quotation marks must be balanced appropriately.
- Incorrect format for date/time data in input files.
- Out-of-range values in input files (for numeric columns).
- Number of distinct values for a column exceeds the limitation for its compression encoding.

Multibyte character load errors

Columns with a CHAR data type only accept single-byte UTF-8 characters, up to byte value 127, or 7F hex, which is also the ASCII character set. VARCHAR columns accept multibyte UTF-8 characters, to a maximum of four bytes. For more information, see [Character types \(p. 582\)](#).

If a line in your load data contains a character that is not valid for the column data type, COPY returns an error and logs a row in the STL_LOAD_ERRORS system log table with error number 1220. The ERR_REASON field includes the byte sequence, in hex, for the invalid character.

An alternative to fixing not valid characters in your load data is to replace the not valid characters during the load process. To replace not valid UTF-8 characters, specify the ACCEPTINVCHARS option with the COPY command. If the ACCEPTINVCHARS option is set, the character you specify replaces the code point. If the ACCEPTINVCHARS option isn't set, Amazon Redshift accepts the characters as valid UTF-8. For more information, see [ACCEPTINVCHARS \(p. 707\)](#).

The following list of code points are valid UTF-8, COPY operations don't return an error if the ACCEPTINVCHARS option is not set. However, these code points are not valid characters. You can use the [ACCEPTINVCHARS \(p. 707\)](#) option to replace a code point with a character that you specify. These code points include the range of values from 0xFDD0 to 0xFDEF and values up to 0x10FFFF, ending with FFFE or FFFF:

- 0xFFFFE, 0x1FFFFE, 0x2FFFFE, ..., 0xFFFFFE, 0x10FFFFE
- 0xFFFFF, 0x1FFFFF, 0x2FFFFF, ..., 0xFFFFFFF, 0x10FFFFF

The following example shows the error reason when COPY attempts to load UTF-8 character e0 a1 c7a4 into a CHAR column.

```
Multibyte character not supported for CHAR
(Hint: Try using VARCHAR). Invalid char: e0 a1 c7a4
```

If the error is related to a VARCHAR data type, the error reason includes an error code as well as the not valid UTF-8 hex sequence. The following example shows the error reason when COPY attempts to load UTF-8 a4 into a VARCHAR field.

```
String contains invalid or unsupported UTF-8 codepoints.  
Bad UTF-8 hex sequence: a4 (error 3)
```

The following table lists the descriptions and suggested workarounds for VARCHAR load errors. If one of these errors occurs, replace the character with a valid UTF-8 code sequence or remove the character.

Error code	Description
1	The UTF-8 byte sequence exceeds the four-byte maximum supported by VARCHAR.
2	The UTF-8 byte sequence is incomplete. COPY did not find the expected number of continuation bytes for a multibyte character before the end of the string.
3	The UTF-8 single-byte character is out of range. The starting byte must not be 254, 255 or any character between 128 and 191 (inclusive).
4	The value of the trailing byte in the byte sequence is out of range. The continuation byte must be between 128 and 191 (inclusive).
5	The UTF-8 character is reserved as a surrogate. Surrogate code points (U+D800 through U+DFFF) are not valid.
8	The byte sequence exceeds the maximum UTF-8 code point.
9	The UTF-8 byte sequence does not have a matching code point.

Load error reference

If any errors occur while loading data from a file, query the [STL_LOAD_ERRORS \(p. 1526\)](#) table to identify the error and determine the possible explanation. The following table lists all error codes that might occur during data loads:

Load error codes

Error code	Description
1200	Unknown parse error. Contact support.
1201	Field delimiter was not found in the input file.
1202	Input data had more columns than were defined in the DDL.
1203	Input data had fewer columns than were defined in the DDL.
1204	Input data exceeded the acceptable range for the data type.
1205	Date format is not valid. See DATEFORMAT and TIMEFORMAT strings (p. 724) for valid formats.
1206	Timestamp format is not valid. See DATEFORMAT and TIMEFORMAT strings (p. 724) for valid formats.
1207	Data contained a value outside of the expected range of 0-9.

Error code	Description
1208	FLOAT data type format error.
1209	DECIMAL data type format error.
1210	BOOLEAN data type format error.
1211	Input line contained no data.
1212	Load file was not found.
1213	A field specified as NOT NULL contained no data.
1214	Delimiter not found.
1215	CHAR field error.
1216	Input line is not valid.
1217	Identity column value is not valid.
1218	When using NULL AS '\0', a field containing a null terminator (NUL, or UTF-8 0000) contained more than one byte.
1219	UTF-8 hexadecimal contains an invalid digit.
1220	String contains invalid or unsupported UTF-8 code points.
1221	Encoding of the file is not the same as that specified in the COPY command.
1222	Integer value overflow error.
1223	Data type not valid.
1224	Input data not well formed JSON format or super data type.
8001	COPY with MANIFEST parameter requires full path of an Amazon S3 object.
9005	Invalid end key specified.

Continuous file ingestion from Amazon S3 (preview)

This is prerelease documentation for auto-copy (SQL COPY JOB), which is in preview release. The documentation and the feature are both subject to change. We recommend that you use this feature only in test environments, and not in production environments. Public preview will end on May 1, 2023. Preview clusters and preview serverless workgroups and namespaces will be removed automatically two weeks after the end of the preview. For preview terms and conditions, see [Betas and Previews in AWS Service Terms](#).

Note

You can create an Amazon Redshift cluster in **Preview** to test new features of Amazon Redshift. You can't use those features in production or move your **Preview** cluster to a production cluster or a cluster on another track. For preview terms and conditions, see *Beta and Previews in [AWS Service Terms](#)*.

To create a cluster in Preview

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Provisioned clusters dashboard**, and choose **Clusters**. The clusters for your account in the current AWS Region are listed. A subset of properties of each cluster is displayed in columns in the list.
3. A banner displays on the **Clusters** list page that introduces preview. Choose the button **Create preview cluster** to open the create cluster page.
4. Enter properties for your cluster. Choose the **Preview track** that contains the features you want to test. We recommend entering a name for the cluster that indicates that it is on a preview track. Choose options for your cluster, including options labeled as **-preview**, for the features you want to test. For general information about creating clusters, see [Creating a cluster](#) in the *Amazon Redshift Management Guide*.
5. Choose **Create preview cluster** to create a cluster in preview.
6. When your preview cluster is available, use your SQL client to load and query data.

Your cluster must be created with the preview track named: preview_2022. Use a new cluster for testing, restoring a cluster into this track is not supported. The autocopy feature is not available with Amazon Redshift Serverless workgroup.

You can create an Amazon Redshift Serverless workgroup in **Preview** to test new features of Amazon Redshift Serverless. You can't use those features in production or move your **Preview** workgroup to a production workgroup. For preview terms and conditions, see *Beta and Previews* in [AWS Service Terms](#).

To create a workgroup in Preview

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Serverless dashboard**, and choose **Workgroup configuration**. The workgroups for your account in the current AWS Region are listed. A subset of properties of each workgroup is displayed in columns in the list.
3. A banner displays on the **Workgroups** list page that introduces preview. Choose the button **Create preview workgroup** to open the create workgroup page.
4. Enter properties for your workgroup. We recommend entering a name for the workgroup that indicates that it is in preview. Choose options for your workgroup, including options labeled as **-preview**, for the features you want to test. Continue through the pages to enter options for your workgroup and namespace. For general information about creating workgroups, see [Creating a workgroup with a namespace](#) in the *Amazon Redshift Management Guide*.
5. Choose **Create preview workgroup** to create a workgroup in preview.
6. When your preview workgroup is available, use your SQL client to load and query data.

This preview is available in the following AWS Regions:

- US East (Ohio) Region (us-east-2)
- US East (N. Virginia) Region (us-east-1)
- US West (Oregon) Region (us-west-2)
- Asia Pacific (Tokyo) Region (ap-northeast-1)
- Europe (Stockholm) Region (eu-north-1)
- Europe (Ireland) Region (eu-west-1)

You can use a COPY JOB to load data into your Amazon Redshift tables from files that are stored in Amazon S3. Amazon Redshift detects when new Amazon S3 files are added to the path specified in your COPY command. A COPY command is then automatically run without you having to create an external data ingestion pipeline. Amazon Redshift keeps track of which files have been loaded. Amazon Redshift determines the number of files batched together per COPY command. You can see the resulting COPY commands in system views.

You define a COPY JOB one time. The same parameters are used for future runs.

You manage the load operations using options to CREATE, LIST, SHOW, DROP, ALTER, and RUN jobs. For more information, see [COPY JOB \(preview\) \(p. 681\)](#).

You can query system views to see the COPY JOB status and progress. Views are provided as follows:

- [SYS_COPY_JOB \(preview\) \(p. 1468\)](#) – contains a row for each currently defined COPY JOB.
- [STL_LOAD_ERRORS \(p. 1526\)](#) – contains errors from COPY commands.
- [STL_LOAD_COMMITS \(p. 1523\)](#) – contains information used to troubleshoot a COPY command data load.
- [SYS_LOAD_HISTORY \(p. 1477\)](#) – contains details of COPY commands.
- [SYS_LOAD_ERROR_DETAIL \(p. 1476\)](#) – contains details of COPY command errors.

To get the list of files loaded by a COPY JOB, run the following example replacing `<job_id>`:

```
SELECT job_id, job_name, data_source, copy_query,filename,status, curtime
FROM sys_copy_job copyjob
JOIN stl_load_commits loadcommit
ON copyjob.job_id = loadcommit.copy_job_id
WHERE job_id = <job_id>;
```

Updating tables with DML commands

Amazon Redshift supports standard data manipulation language (DML) commands (INSERT, UPDATE, and DELETE) that you can use to modify rows in tables. You can also use the TRUNCATE command to do fast bulk deletes.

Note

We strongly encourage you to use the [COPY \(p. 677\)](#) command to load large amounts of data. Using individual INSERT statements to populate a table might be prohibitively slow. Alternatively, if your data already exists in other Amazon Redshift database tables, use INSERT INTO ... SELECT FROM or CREATE TABLE AS to improve performance. For information, see [INSERT \(p. 902\)](#) or [CREATE TABLE AS \(p. 838\)](#).

If you insert, update, or delete a significant number of rows in a table, relative to the number of rows before the changes, run the ANALYZE and VACUUM commands against the table when you are done. If a number of small changes accumulate over time in your application, you might want to schedule the ANALYZE and VACUUM commands to run at regular intervals. For more information, see [Analyzing tables \(p. 105\)](#) and [Vacuuming tables \(p. 110\)](#).

Updating and inserting new data

You can efficiently add new data to an existing table by using a combination of updates and inserts from a staging table. While Amazon Redshift does not support a single *merge*, or *upsert*, command to update a

table from a single data source, you can perform a merge operation by creating a staging table and then using one of the methods described in this section to update the target table from the staging table.

Topics

- [Merge method 1: Replacing existing rows \(p. 97\)](#)
- [Merge method 2: Specifying a column list \(p. 97\)](#)
- [Creating a temporary staging table \(p. 97\)](#)
- [Performing a merge operation by replacing existing rows \(p. 98\)](#)
- [Performing a merge operation by specifying a column list \(p. 99\)](#)
- [Merge examples \(p. 100\)](#)

The [Merge examples \(p. 100\)](#) use a sample dataset for Amazon Redshift, called the TICKIT data set. As a prerequisite, you can set up the TICKIT tables and data by following the instructions available in [Getting started with common database tasks](#). More detailed information about the sample data set is found at [Sample database](#).

Merge method 1: Replacing existing rows

If you are overwriting all of the columns in the target table, the fastest method to perform a merge is to replace the existing rows. This scans the target table only once, by using an inner join to delete rows that will be updated. After the rows are deleted, they are replaced with new rows by a single insert operation from the staging table.

Use this method if all of the following are true:

- Your target table and your staging table contain the same columns.
- You intend to replace all of the data in the target table columns with all of the staging table columns.
- You will use all of the rows in the staging table in the merge.

If any of these criteria do not apply, use Merge method 2: Specifying a column list, described in the following section.

If you will not use all of the rows in the staging table, filter the DELETE and INSERT statements by using a WHERE clause to leave out rows that are not changing. However, if most of the rows in the staging table will not participate in the merge, we recommend performing an UPDATE and an INSERT in separate steps, as described later in this section.

Merge method 2: Specifying a column list

Use this method to update specific columns in the target table instead of overwriting entire rows. This method takes longer than the previous method because it requires an extra update step. Use this method if any of the following are true:

- Not all of the columns in the target table are to be updated.
- Most rows in the staging table will not be used in the updates.

Creating a temporary staging table

The *staging table* is a temporary table that holds all of the data that will be used to make changes to the *target table*, including both updates and inserts.

A merge operation requires a join between the staging table and the target table. To collocate the joining rows, set the staging table's distribution key to the same column as the target table's distribution key. For example, if the target table uses a foreign key column as its distribution key, use the same column for the staging table's distribution key. If you create the staging table by using a [CREATE TABLE LIKE \(p. 828\)](#) statement, the staging table will inherit the distribution key from the parent table. If you use a CREATE TABLE AS statement, the new table does not inherit the distribution key. For more information, see [Working with data distribution styles \(p. 49\)](#)

If the distribution key is not the same as the primary key and the distribution key is not updated as part of the merge operation, add a redundant join predicate on the distribution key columns to enable a collocated join. For example:

```
where target.primarykey = stage.primarykey  
and target.distkey = stage.distkey
```

To verify that the query will use a collocated join, run the query with [EXPLAIN \(p. 881\)](#) and check for DS_DIST_NONE on all of the joins. For more information, see [Evaluating the query plan \(p. 54\)](#)

Performing a merge operation by replacing existing rows

When you run the merge operation detailed in the procedure, put all of the steps except for creating and dropping the temporary staging table in a single transaction. The transaction rolls back if any step fails. Using a single transaction also reduces the number of commits, which saves time and resources.

To perform a merge operation by replacing existing rows

1. Create a staging table, and then populate it with data to be merged, as shown in the following pseudocode.

```
create temp table stage (like target);  
  
insert into stage  
select * from source  
where source.filter = 'filter_expression';
```

2. Use an inner join with the staging table to delete the rows from the target table that are being updated.

Put the delete and insert operations in a single transaction block so that if there is a problem, everything will be rolled back.

```
begin transaction;  
  
delete from target  
using stage  
where target.primarykey = stage.primarykey;
```

3. Insert all of the rows from the staging table.

```
insert into target  
select * from stage;  
  
end transaction;
```

4. Drop the staging table.

```
drop table stage;
```

Performing a merge operation by specifying a column list

When you run the merge operation detailed in the procedure, put all of the steps in a single transaction. The transaction rolls back if any step fails. Using a single transaction also reduces the number of commits, which saves time and resources.

To perform a merge operation by specifying a column list

1. Put the entire operation in a single transaction block.

```
begin transaction;  
...  
end transaction;
```

2. Create a staging table, and then populate it with data to be merged, as shown in the following pseudocode.

```
create temp table stage (like target);  
insert into stage  
select * from source  
where source.filter = 'filter_expression';
```

3. Update the target table by using an inner join with the staging table.

- In the UPDATE clause, explicitly list the columns to be updated.
- Perform an inner join with the staging table.
- If the distribution key is different from the primary key and the distribution key is not being updated, add a redundant join on the distribution key. To verify that the query will use a collocated join, run the query with [EXPLAIN \(p. 881\)](#) and check for DS_DIST_NONE on all of the joins. For more information, see [Evaluating the query plan \(p. 54\)](#).
- If your target table is sorted by timestamp, add a predicate to take advantage of range-restricted scans on the target table. For more information, see [Amazon Redshift best practices for designing queries \(p. 21\)](#).
- If you will not use all of the rows in the merge, add a clause to filter the rows that you want to change. For example, add an inequality filter on one or more columns to exclude rows that have not changed.
- Put the update, delete, and insert operations in a single transaction block so that if there is a problem, everything will be rolled back.

For example:

```
begin transaction;  
  
update target  
set col1 = stage.col1,  
    col2 = stage.col2,  
    col3 = 'expression'  
from stage  
where target.primarykey = stage.primarykey  
and target.distkey = stage.distkey
```

```
and target.col3 > 'last_update_time'  
and (target.col1 != stage.col1  
or target.col2 != stage.col2  
or target.col3 = 'filter_expression');
```

4. Delete unneeded rows from the staging table by using an inner join with the target table. Some rows in the target table already match the corresponding rows in the staging table, and others were updated in the previous step. In either case, they are not needed for the insert.

```
delete from stage  
using target  
where stage.primarykey = target.primarykey;
```

5. Insert the remaining rows from the staging table. Use the same column list in the VALUES clause that you used in the UPDATE statement in step two.

```
insert into target  
(select col1, col2, 'expression'  
from stage);  
  
end transaction;
```

6. Drop the staging table.

```
drop table stage;
```

Merge examples

The following examples perform a merge to update the SALES table. The first example uses the simpler method of deleting from the target table and then inserting all of the rows from the staging table. The second example requires updating on select columns in the target table, so it includes an extra update step.

The [Merge examples \(p. 100\)](#) use a sample dataset for Amazon Redshift, called the TICKIT data set. As a prerequisite, you can set up the TICKIT tables and data by following the instructions available in the guide [Getting started with common database tasks](#). More detailed information about the sample data set is found at [Sample database](#).

Sample merge data source

The examples in this section need a sample data source that includes both updates and inserts. For the examples, we will create a sample table named SALES_UPDATE that uses data from the SALES table. We'll populate the new table with random data that represents new sales activity for December. We will use the SALES_UPDATE sample table to create the staging table in the examples that follow.

```
-- Create a sample table as a copy of the SALES table  
create table sales_update as  
select * from sales;  
  
-- Change every fifth row so we have updates  
  
update sales_update  
set qtysold = qtysold*2,  
pricepaid = pricepaid*0.8,  
commission = commission*1.1  
where saletime > '2008-11-30'  
and mod(sellerid, 5) = 0;
```

```
-- Add some new rows so we have insert examples
-- This example creates a duplicate of every fourth row

insert into sales_update
select (salesid + 172456) as salesid, listid, sellerid, buyerid, eventid, dateid, qtysold,
       pricepaid, commission, getdate() as saletime
  from sales_update
 where saletime > '2008-11-30'
   and mod(sellerid, 4) = 0;
```

Example of a merge that replaces existing rows

The following script uses the SALES_UPDATE table to perform a merge operation on the SALES table with new data for December sales activity. This example deletes rows in the SALES table that have updates so they can be replaced with the updated rows in the staging table. The staging table should contain only rows that will participate in the merge, so the CREATE TABLE statement includes a filter to exclude rows that have not changed.

```
-- Create a staging table and populate it with updated rows from SALES_UPDATE

create temp table stagesales as
select * from sales_update
where sales_update.saletime > '2008-11-30'
and sales_update.salesid = (select sales.salesid from sales
                           where sales.salesid = sales_update.salesid
                           and sales.listid = sales_update.listid
                           and (sales_update.qtysold != sales.qtysold
                                or sales_update.pricepaid != sales.pricepaid));

-- Start a new transaction
begin transaction;

-- Delete any rows from SALES that exist in STAGESALES, because they are updates
-- The join includes a redundant predicate to collocate on the distribution key
-- A filter on saletime enables a range-restricted scan on SALES

delete from sales
using stagesales
where sales.salesid = stagesales.salesid
and sales.listid = stagesales.listid
and sales.saletime > '2008-11-30';

-- Insert all the rows from the staging table into the target table
insert into sales
select * from stagesales;

-- End transaction and commit
end transaction;

-- Drop the staging table
drop table stagesales;
```

Example of a merge that specifies a column list

The following example performs a merge operation to update SALES with new data for December sales activity. We need sample data that includes both updates and inserts, along with rows that have not changed. For this example, we want to update the QTYSOLD and PRICEPAID columns but leave COMMISSION and SALETIME unchanged. The following script uses the SALES_UPDATE table to perform a merge operation on the SALES table.

```
-- Create a staging table and populate it with rows from SALES_UPDATE for Dec
create temp table stagesales as select * from sales_update
```

```
where saletime > '2008-11-30';

-- Start a new transaction
begin transaction;

-- Update the target table using an inner join with the staging table
-- The join includes a redundant predicate to collocate on the distribution key -- A filter
on saletime enables a range-restricted scan on SALES

update sales
set qtysold = stagesales.qtysold,
pricepaid = stagesales.pricepaid
from stagesales
where sales.salesid = stagesales.salesid
and sales.listid = stagesales.listid
and stagesales.saletime > '2008-11-30'
and (sales.qtysold != stagesales.qtysold
or sales.pricepaid != stagesales.pricepaid);

-- Delete matching rows from the staging table
-- using an inner join with the target table

delete from stagesales
using sales
where sales.salesid = stagesales.salesid
and sales.listid = stagesales.listid;

-- Insert the remaining rows from the staging table into the target table
insert into sales
select * from stagesales;

-- End transaction and commit
end transaction;

-- Drop the staging table
drop table stagesales;
```

Performing a deep copy

A deep copy recreates and repopulates a table by using a bulk insert, which automatically sorts the table. If a table has a large unsorted Region, a deep copy is much faster than a vacuum. We recommend that you only make concurrent updates during a deep copy operation if you can track them. After the process has completed, move the delta updates into the new table. A VACUUM operation supports concurrent updates automatically.

You can choose one of the following methods to create a copy of the original table:

- Use the original table DDL.

If the CREATE TABLE DDL is available, this is the fastest and preferred method. If you create a new table, you can specify all table and column attributes, including primary key and foreign keys. You can find the original DDL by using the SHOW TABLE function.

- Use CREATE TABLE LIKE.

If the original DDL is not available, you can use CREATE TABLE LIKE to recreate the original table. The new table inherits the encoding, distribution key, sort key, and not-null attributes of the parent table. The new table doesn't inherit the primary key and foreign key attributes of the parent table, but you can add them using [ALTER TABLE \(p. 640\)](#).

- Create a temporary table and truncate the original table.

If you must retain the primary key and foreign key attributes of the parent table. If the parent table has dependencies, you can use CREATE TABLE ... AS (CTAS) to create a temporary table. Then truncate the original table and populate it from the temporary table.

Using a temporary table improves performance significantly compared to using a permanent table, but there is a risk of losing data. A temporary table is automatically dropped at the end of the session in which it is created. TRUNCATE commits immediately, even if it is inside a transaction block. If the TRUNCATE succeeds but the session shuts down before the following INSERT completes, the data is lost. If data loss is unacceptable, use a permanent table.

After you create a copy of a table, you might have to grant access to the new table. You can use [GRANT \(p. 887\)](#) to define access privileges. To view and grant all of a table's access privileges, you must be one of the following:

- A superuser.
- The owner of the table you want to copy.
- A user with the ACCESS SYSTEM TABLE privilege to see the table's privileges, and with the grant privilege for all relevant permissions.

Additionally, you might have to grant usage permission for the schema your deep copy is in. Granting usage permission is necessary if your deep copy's schema is different from the original table's schema, and also isn't the public schema. To view and grant usage privileges you must be one of the following:

- A superuser.
- A user who can grant the USAGE permission for the deep copy's schema.

To perform a deep copy using the original table DDL

1. (Optional) Recreate the table DDL by running a script called v_generate_tbl_ddl.
2. Create a copy of the table using the original CREATE TABLE DDL.
3. Use an INSERT INTO ... SELECT statement to populate the copy with data from the original table.
4. Check for permissions granted on the old table. You can see these permissions in the SVV_RELATION_PRIVILEGES system view.
5. If necessary, grant the permissions of the old table to the new table.
6. Grant usage permission to every group and user that has privileges in the original table. This step isn't necessary if your deep copy table is in the public schema, or is in the same schema as the original table.
7. Drop the original table.
8. Use an ALTER TABLE statement to rename the copy to the original table name.

The following example performs a deep copy on the SAMPLE table using a duplicate of SAMPLE named sample_copy.

```
--Create a copy of the original table in the sample_namespace namespace using the original
CREATE TABLE DDL.
create table sample_namespace.sample_copy ( ... );

--Populate the copy with data from the original table in the public namespace.
insert into sample_namespace.sample_copy (select * from public.sample);

--Check SVV_RELATION_PRIVILEGES for the original table's privileges.
```

```

select * from svv_relation_privileges where namespace_name = 'public' and relation_name =
'sample' order by identity_type, identity_id, privilege_type;

--Grant the original table's privileges to the copy table.
grant DELETE on table sample_namespace.sample_copy to group group1;
grant INSERT, UPDATE on table sample_namespace.sample_copy to group group2;
grant SELECT on table sample_namespace.sample_copy to user1;
grant INSERT, SELECT, UPDATE on table sample_namespace.sample_copy to user2;

--Grant usage permission to every group and user that has privileges in the original table.
grant USAGE on schema sample_namespace to group group1, group group2, user1, user2;

--Drop the original table.
drop table public.sample;

--Rename the copy table to match the original table's name.
alter table sample_namespace.sample_copy rename to sample;

```

To perform a deep copy using CREATE TABLE LIKE

1. Create a new table using CREATE TABLE LIKE.
2. Use an INSERT INTO ... SELECT statement to copy the rows from the current table to the new table.
3. Check for permissions granted on the old table. You can see these permissions in the SVV_RELATION_PRIVILEGES system view.
4. If necessary, grant the permissions of the old table to the new table.
5. Grant usage permission to every group and user that has privileges in the original table. This step isn't necessary if your deep copy table is in the public schema, or is in the same schema as the original table.
6. Drop the current table.
7. Use an ALTER TABLE statement to rename the new table to the original table name.

The following example performs a deep copy on the SAMPLE table using CREATE TABLE LIKE.

```

--Create a copy of the original table in the sample_namespace namespace using CREATE TABLE
--LIKE.
create table sample_namespace.sample_copy (like public.sample);

--Populate the copy with data from the original table.
insert into sample_namespace.sample_copy (select * from public.sample);

--Check SVV_RELATION_PRIVILEGES for the original table's privileges.
select * from svv_relation_privileges where namespace_name = 'public' and relation_name =
'sample' order by identity_type, identity_id, privilege_type;

--Grant the original table's privileges to the copy table.
grant DELETE on table sample_namespace.sample_copy to group group1;
grant INSERT, UPDATE on table sample_namespace.sample_copy to group group2;
grant SELECT on table sample_namespace.sample_copy to user1;
grant INSERT, SELECT, UPDATE on table sample_namespace.sample_copy to user2;

--Grant usage permission to every group and user that has privileges in the original table.
grant USAGE on schema sample_namespace to group group1, group group2, user1, user2;

--Drop the original table.
drop table public.sample;

--Rename the copy table to match the original table's name.
alter table sample_namespace.sample_copy rename to sample;

```

To perform a deep copy by creating a temporary table and truncating the original table

1. Use CREATE TABLE AS to create a temporary table with the rows from the original table.
2. Truncate the current table.
3. Use an INSERT INTO ... SELECT statement to copy the rows from the temporary table to the original table.
4. Drop the temporary table.

The following example performs a deep copy on the SALES table by creating a temporary table and truncating the original table. Since the original table remains, you don't need to grant permissions to the copy table.

```
--Create a temp table copy using CREATE TABLE AS.  
create temp table salestemp as select * from sales;  
  
--Truncate the original table.  
truncate sales;  
  
--Copy the rows from the temporary table to the original table.  
insert into sales (select * from salestemp);  
  
--Drop the temporary table.  
drop table salestemp;
```

Analyzing tables

The ANALYZE operation updates the statistical metadata that the query planner uses to choose optimal plans.

In most cases, you don't need to explicitly run the ANALYZE command. Amazon Redshift monitors changes to your workload and automatically updates statistics in the background. In addition, the COPY command performs an analysis automatically when it loads data into an empty table.

To explicitly analyze a table or the entire database, run the [ANALYZE \(p. 663\)](#) command.

Topics

- [Automatic analyze \(p. 105\)](#)
- [Analysis of new table data \(p. 106\)](#)
- [ANALYZE command history \(p. 109\)](#)

Automatic analyze

Amazon Redshift continuously monitors your database and automatically performs analyze operations in the background. To minimize impact to your system performance, automatic analyze runs during periods when workloads are light.

Automatic analyze is enabled by default. To turn off automatic analyze, set the `auto_analyze` parameter to `false` by modifying your cluster's parameter group.

To reduce processing time and improve overall system performance, Amazon Redshift skips automatic analyze for any table where the extent of modifications is small.

An analyze operation skips tables that have up-to-date statistics. If you run ANALYZE as part of your extract, transform, and load (ETL) workflow, automatic analyze skips tables that have current statistics. Similarly, an explicit ANALYZE skips tables when automatic analyze has updated the table's statistics.

Analysis of new table data

By default, the COPY command performs an ANALYZE after it loads data into an empty table. You can force an ANALYZE regardless of whether a table is empty by setting STATUPDATE ON. If you specify STATUPDATE OFF, an ANALYZE is not performed. Only the table owner or a superuser can run the ANALYZE command or run the COPY command with STATUPDATE set to ON.

Amazon Redshift also analyzes new tables that you create with the following commands:

- CREATE TABLE AS (CTAS)
- CREATE TEMP TABLE AS
- SELECT INTO

Amazon Redshift returns a warning message when you run a query against a new table that was not analyzed after its data was initially loaded. No warning occurs when you query a table after a subsequent update or load. The same warning message is returned when you run the EXPLAIN command on a query that references tables that have not been analyzed.

Whenever adding data to a nonempty table significantly changes the size of the table, you can explicitly update statistics. You do so either by running an ANALYZE command or by using the STATUPDATE ON option with the COPY command. To view details about the number of rows that have been inserted or deleted since the last ANALYZE, query the [PG_STATISTIC_INDICATOR \(p. 1676\)](#) system catalog table.

You can specify the scope of the [ANALYZE \(p. 663\)](#) command to one of the following:

- The entire current database
- A single table
- One or more specific columns in a single table
- Columns that are likely to be used as predicates in queries

The ANALYZE command gets a sample of rows from the table, does some calculations, and saves resulting column statistics. By default, Amazon Redshift runs a sample pass for the DISTKEY column and another sample pass for all of the other columns in the table. If you want to generate statistics for a subset of columns, you can specify a comma-separated column list. You can run ANALYZE with the PREDICATE COLUMNS clause to skip columns that aren't used as predicates.

ANALYZE operations are resource intensive, so run them only on tables and columns that actually require statistics updates. You don't need to analyze all columns in all tables regularly or on the same schedule. If the data changes substantially, analyze the columns that are frequently used in the following:

- Sorting and grouping operations
- Joins
- Query predicates

To reduce processing time and improve overall system performance, Amazon Redshift skips ANALYZE for any table that has a low percentage of changed rows, as determined by the [analyze_threshold_percent \(p. 1685\)](#) parameter. By default, the analyze threshold is set to 10 percent. You can change the analyze threshold for the current session by running a [SET \(p. 977\)](#) command.

Columns that are less likely to require frequent analysis are those that represent facts and measures and any related attributes that are never actually queried, such as large VARCHAR columns. For example, consider the LISTING table in the TICKIT database.

```
select "column", type, encoding, distkey, sortkey
```

```
from pg_table_def where tablename = 'listing';

column          |      type       | encoding | distkey | sortkey
-----+-----+-----+-----+-----+
listid          | integer        | none     | t       | 1
sellerid        | integer        | none     | f       | 0
eventid         | integer        | mostly16| f       | 0
dateid          | smallint       | none     | f       | 0
numtickets      | smallint       | mostly8  | f       | 0
priceperticket | numeric(8,2)   | bytedict | f       | 0
totalprice      | numeric(8,2)   | mostly32 | f       | 0
listtime        | timestamp with... | none     | f       | 0
```

If this table is loaded every day with a large number of new records, the LISTID column, which is frequently used in queries as a join key, must be analyzed regularly. If TOTALPRICE and LISTTIME are the frequently used constraints in queries, you can analyze those columns and the distribution key on every weekday.

```
analyze listing(listid, totalprice, listtime);
```

Suppose that the sellers and events in the application are much more static, and the date IDs refer to a fixed set of days covering only two or three years. In this case, the unique values for these columns don't change significantly. However, the number of instances of each unique value will increase steadily.

In addition, consider the case where the NUMTICKETS and PRICEPERTICKET measures are queried infrequently compared to the TOTALPRICE column. In this case, you can run the ANALYZE command on the whole table once every weekend to update statistics for the five columns that are not analyzed daily:

Predicate columns

As a convenient alternative to specifying a column list, you can choose to analyze only the columns that are likely to be used as predicates. When you run a query, any columns that are used in a join, filter condition, or group by clause are marked as predicate columns in the system catalog. When you run ANALYZE with the PREDICATE COLUMNS clause, the analyze operation includes only columns that meet the following criteria:

- The column is marked as a predicate column.
 - The column is a distribution key.
 - The column is part of a sort key.

If none of a table's columns are marked as predicates, ANALYZE includes all of the columns, even when PREDICATE COLUMNS is specified. If no columns are marked as predicate columns, it might be because the table has not yet been queried.

You might choose to use PREDICATE COLUMNS when your workload's query pattern is relatively stable. When the query pattern is variable, with different columns frequently being used as predicates, using PREDICATE COLUMNS might temporarily result in stale statistics. Stale statistics can lead to suboptimal query runtime plans and long runtimes. However, the next time you run ANALYZE using PREDICATE COLUMNS, the new predicate columns are included.

To view details for predicate columns, use the following SQL to create a view named PREDICATE_COLUMNS.

```
CREATE VIEW predicate_columns AS
WITH predicate_column_info AS (
SELECT ns.nspname AS schema_name, c.relname AS table_name, a.attnum as col_num,  a.attname
as col_name,
CASE
    WHEN 10002 = s.stakind1 THEN array_to_string(stavalue1, '|||')
```

```

WHEN 10002 = s.stakind2 THEN array_to_string(stavalues2, '||')
WHEN 10002 = s.stakind3 THEN array_to_string(stavalues3, '||')
WHEN 10002 = s.stakind4 THEN array_to_string(stavalues4, '||')
ELSE NULL::varchar
END AS pred_ts
FROM pg_statistic s
JOIN pg_class c ON c.oid = s.starelid
JOIN pg_namespace ns ON c.relnamespace = ns.oid
JOIN pg_attribute a ON c.oid = a.attrelid AND a.attnum = s.staattnum
SELECT schema_name, table_name, col_num, col_name,
pred_ts NOT LIKE '2000-01-01%' AS is_predicate,
CASE WHEN pred_ts NOT LIKE '2000-01-01%' THEN (split_part(pred_ts,
'||',1))::timestamp ELSE NULL::timestamp END as first_predicate_use,
CASE WHEN pred_ts NOT LIKE '%||2000-01-01%' THEN (split_part(pred_ts,
'||',2))::timestamp ELSE NULL::timestamp END as last_analyze
FROM predicate_column_info;

```

Suppose that you run the following query against the LISTING table. Note that LISTID, LISTTIME, and EVENTID are used in the join, filter, and group by clauses.

```

select s.buyerid,l.eventid, sum(l.totalprice)
from listing l
join sales s on l.listid = s.listid
where l.listtime > '2008-12-01'
group by l.eventid, s.buyerid;

```

When you query the PREDICATE_COLUMNS view, as shown in the following example, you see that LISTID, EVENTID, and LISTTIME are marked as predicate columns.

```

select * from predicate_columns
where table_name = 'listing';

```

schema_name	table_name	col_num	col_name	is_predicate	first_predicate_use	last_analyze
public	listing	1	listid	true	2017-05-05 19:27:59	2017-05-03 18:27:41
public	listing	2	sellerid	false		2017-05-03 18:27:41
public	listing	3	eventid	true	2017-05-16 20:54:32	2017-05-03 18:27:41
public	listing	4	dateid	false		2017-05-03 18:27:41
public	listing	5	numtickets	false		2017-05-03 18:27:41
public	listing	6	priceperticket	false		2017-05-03 18:27:41
public	listing	7	totalprice	false		2017-05-03 18:27:41
public	listing	8	listtime	true	2017-05-16 20:54:32	2017-05-03 18:27:41

Keeping statistics current improves query performance by enabling the query planner to choose optimal plans. Amazon Redshift refreshes statistics automatically in the background, and you can also explicitly run the ANALYZE command. If you choose to explicitly run ANALYZE, do the following:

- Run the ANALYZE command before running queries.
- Run the ANALYZE command on the database routinely at the end of every regular load or update cycle.

- Run the ANALYZE command on any new tables that you create and any existing tables or columns that undergo significant change.
- Consider running ANALYZE operations on different schedules for different types of tables and columns, depending on their use in queries and their propensity to change.
- To save time and cluster resources, use the PREDICATE COLUMNS clause when you run ANALYZE.

An analyze operation skips tables that have up-to-date statistics. If you run ANALYZE as part of your extract, transform, and load (ETL) workflow, automatic analyze skips tables that have current statistics. Similarly, an explicit ANALYZE skips tables when automatic analyze has updated the table's statistics.

ANALYZE command history

It's useful to know when the last ANALYZE command was run on a table or database. When an ANALYZE command is run, Amazon Redshift runs multiple queries that look like this:

```
padb_fetch_sample: select * from table_name
```

Query STL_ANALYZE to view the history of analyze operations. If Amazon Redshift analyzes a table using automatic analyze, the `is_background` column is set to t (true). Otherwise, it is set to f (false). The following example joins STV_TBL_PERM to show the table name and runtime details.

```
select distinct a.xid, trim(t.name) as name, a.status, a.rows, a.modified_rows,
a starttime, a.endtime
from stl_analyze a
join stv_tbl_perm t on t.id=a.table_id
where name = 'users'
order by starttime;

xid      | name    | status        | rows   | modified_rows | starttime           | endtime
-----+-----+-----+-----+-----+-----+-----+
1582 | users | Full          | 49990 | 49990 | 2016-09-22 22:02:23 | 2016-09-22
22:02:28
244287 | users | Full          | 24992 | 74988 | 2016-10-04 22:50:58 | 2016-10-04
22:51:01
244712 | users | Full          | 49984 | 24992 | 2016-10-04 22:56:07 | 2016-10-04
22:56:07
245071 | users | Skipped       | 49984 | 0      | 2016-10-04 22:58:17 | 2016-10-04
22:58:17
245439 | users | Skipped       | 49984 | 1982  | 2016-10-04 23:00:13 | 2016-10-04
23:00:13
(5 rows)
```

Alternatively, you can run a more complex query that returns all the statements that ran in every completed transaction that included an ANALYZE command:

```
select xid, to_char(starttime, 'HH24:MM:SS.MS') as starttime,
datediff(sec,starttime,endtime ) as secs, substring(text, 1, 40)
from svl_statementtext
where sequence = 0
and xid in (select xid from svl_statementtext s where s.text like 'padb_fetch_sample%' )
order by xid desc, starttime;

xid      | starttime | secs | substring
-----+-----+-----+-----+
1338 | 12:04:28.511 | 4 | Analyze date
1338 | 12:04:28.511 | 1 | padb_fetch_sample: select count(*) from
```

```

1338 | 12:04:29.443 |    2 | padb_fetch_sample: select * from date
1338 | 12:04:31.456 |    1 | padb_fetch_sample: select * from date
1337 | 12:04:24.388 |    1 | padb_fetch_sample: select count(*) from
1337 | 12:04:24.388 |    4 | Analyze sales
1337 | 12:04:25.322 |    2 | padb_fetch_sample: select * from sales
1337 | 12:04:27.363 |    1 | padb_fetch_sample: select * from sales
...

```

Vacuuming tables

Amazon Redshift can automatically sort and perform a VACUUM DELETE operation on tables in the background. To clean up tables after a load or a series of incremental updates, you can also run the [VACUUM \(p. 1015\)](#) command, either against the entire database or against individual tables.

Note

Only the table owner or a superuser can effectively vacuum a table. If you don't have owner or superuser permissions for a table, a VACUUM operation that specifies a single table fails. If you run a VACUUM of the entire database without specifying a table name, the operation completes successfully. However, the operation has no effect on tables for which you don't have owner or superuser permissions.

For this reason, we recommend vacuuming individual tables as needed. We also recommend this approach because vacuuming the entire database is potentially an expensive operation.

Automatic table sort

Amazon Redshift automatically sorts data in the background to maintain table data in the order of its sort key. Amazon Redshift keeps track of your scan queries to determine which sections of the table will benefit from sorting.

Depending on the load on the system, Amazon Redshift automatically initiates the sort. This automatic sort lessens the need to run the VACUUM command to keep data in sort key order. If you need data fully sorted in sort key order, for example after a large data load, then you can still manually run the VACUUM command. To determine whether your table will benefit by running VACUUM SORT, monitor the `vacuum_sort_benefit` column in [SVV_TABLE_INFO \(p. 1457\)](#).

Amazon Redshift tracks scan queries that use the sort key on each table. Amazon Redshift estimates the maximum percentage of improvement in scanning and filtering of data for each table (if the table was fully sorted). This estimate is visible in the `vacuum_sort_benefit` column in [SVV_TABLE_INFO \(p. 1457\)](#). You can use this column, along with the `unsorted` column, to determine when queries can benefit from manually running VACUUM SORT on a table. The `unsorted` column reflects the physical sort order of a table. The `vacuum_sort_benefit` column specifies the impact of sorting a table by manually running VACUUM SORT.

For example, consider the following query:

```
select "table", unsorted,vacuum_sort_benefit from svv_table_info order by 1;
```

table	unsorted	vacuum_sort_benefit
sales	85.71	5.00
event	45.24	67.00

For the table “sales”, even though the table is ~86% physically unsorted, the query performance impact from the table being 86% unsorted is only 5%. This might be either because only a small portion of the table is accessed by queries, or very few queries accessed the table. For the table “event”, the table is ~45% physically unsorted. But the query performance impact of 67% indicates that either a larger

portion of the table was accessed by queries, or the number of queries accessing the table was large. The table "event" can potentially benefit from running VACUUM SORT.

Automatic vacuum delete

When you perform a delete, the rows are marked for deletion, but not removed. Amazon Redshift automatically runs a VACUUM DELETE operation in the background based on the number of deleted rows in database tables. Amazon Redshift schedules the VACUUM DELETE to run during periods of reduced load and pauses the operation during periods of high load.

Topics

- [VACUUM frequency \(p. 111\)](#)
- [Sort stage and merge stage \(p. 111\)](#)
- [Vacuum threshold \(p. 112\)](#)
- [Vacuum types \(p. 112\)](#)
- [Managing vacuum times \(p. 112\)](#)

VACUUM frequency

You should vacuum as often as necessary to maintain consistent query performance. Consider these factors when determining how often to run your VACUUM command:

- Run VACUUM during time periods when you expect minimal activity on the cluster, such as evenings or during designated database administration windows.
- A large unsorted region results in longer vacuum times. If you delay vacuuming, the vacuum will take longer because more data has to be reorganized.
- VACUUM is an I/O intensive operation, so the longer it takes for your vacuum to complete, the more impact it will have on concurrent queries and other database operations running on your cluster.
- VACUUM takes longer for tables that use interleaved sorting. To evaluate whether interleaved tables must be re-sorted, query the [SVV_INTERLEAVED_COLUMNS \(p. 1434\)](#) view.

Sort stage and merge stage

Amazon Redshift performs a vacuum operation in two stages: first, it sorts the rows in the unsorted region, then, if necessary, it merges the newly sorted rows at the end of the table with the existing rows. When vacuuming a large table, the vacuum operation proceeds in a series of steps consisting of incremental sorts followed by merges. If the operation fails or if Amazon Redshift goes offline during the vacuum, the partially vacuumed table or database will be in a consistent state, but you must manually restart the vacuum operation. Incremental sorts are lost, but merged rows that were committed before the failure do not need to be vacuumed again. If the unsorted region is large, the lost time might be significant. For more information about the sort and merge stages, see [Managing the volume of merged rows \(p. 113\)](#).

Users can access tables while they are being vacuumed. You can perform queries and write operations while a table is being vacuumed, but when DML and a vacuum run concurrently, both might take longer. If you run UPDATE and DELETE statements during a vacuum, system performance might be reduced. Incremental merges temporarily block concurrent UPDATE and DELETE operations, and UPDATE and DELETE operations in turn temporarily block incremental merge steps on the affected tables. DDL operations, such as ALTER TABLE, are blocked until the vacuum operation finishes with the table.

Note

Various modifiers to VACUUM control the way that it works. You can use them to tailor the vacuum operation for the current need. For example, using VACUUM RECLUSTER shortens

the vacuum operation by not performing a full merge operation. For more information, see [VACUUM \(p. 1015\)](#).

Vacuum threshold

By default, VACUUM skips the sort phase for any table where more than 95 percent of the table's rows are already sorted. Skipping the sort phase can significantly improve VACUUM performance. To change the default sort threshold for a single table, include the table name and the TO *threshold* PERCENT parameter when you run the VACUUM command.

Vacuum types

For information about different vacuum types, see [VACUUM \(p. 1015\)](#).

Managing vacuum times

Depending on the nature of your data, we recommend following the practices in this section to minimize vacuum times.

Topics

- [Deciding whether to reindex \(p. 112\)](#)
- [Managing the size of the unsorted region \(p. 113\)](#)
- [Managing the volume of merged rows \(p. 113\)](#)
- [Loading your data in sort key order \(p. 117\)](#)
- [Using time series tables \(p. 117\)](#)

Deciding whether to reindex

You can often significantly improve query performance by using an interleaved sort style, but over time performance might degrade if the distribution of the values in the sort key columns changes.

When you initially load an empty interleaved table using COPY or CREATE TABLE AS, Amazon Redshift automatically builds the interleaved index. If you initially load an interleaved table using INSERT, you need to run VACUUM REINDEX afterwards to initialize the interleaved index.

Over time, as you add rows with new sort key values, performance might degrade if the distribution of the values in the sort key columns changes. If your new rows fall primarily within the range of existing sort key values, you don't need to reindex. Run VACUUM SORT ONLY or VACUUM FULL to restore the sort order.

The query engine is able to use sort order to efficiently select which data blocks need to be scanned to process a query. For an interleaved sort, Amazon Redshift analyzes the sort key column values to determine the optimal sort order. If the distribution of key values changes, or skews, as rows are added, the sort strategy will no longer be optimal, and the performance benefit of sorting will degrade. To reanalyze the sort key distribution you can run a VACUUM REINDEX. The reindex operation is time consuming, so to decide whether a table will benefit from a reindex, query the [SVV_INTERLEAVED_COLUMNS \(p. 1434\)](#) view.

For example, the following query shows details for tables that use interleaved sort keys.

```
select tbl as tbl_id, stv_tbl_perm.name as table_name,
col, interleaved_skew, last_reindex
from svv_interleaved_columns, stv_tbl_perm
where svv_interleaved_columns.tbl = stv_tbl_perm.id
and interleaved_skew is not null;
```

tbl_id	table_name	col	interleaved_skew	last_reindex
100048	customer	0	3.65	2015-04-22 22:05:45
100068	lineorder	1	2.65	2015-04-22 22:05:45
100072	part	0	1.65	2015-04-22 22:05:45
100077	supplier	1	1.00	2015-04-22 22:05:45
(4 rows)				

The value for `interleaved_skew` is a ratio that indicates the amount of skew. A value of 1 means that there is no skew. If the skew is greater than 1.4, a VACUUM REINDEX will usually improve performance unless the skew is inherent in the underlying set.

You can use the date value in `last_reindex` to determine how long it has been since the last reindex.

Managing the size of the unsorted region

The unsorted region grows when you load large amounts of new data into tables that already contain data or when you do not vacuum tables as part of your routine maintenance operations. To avoid long-running vacuum operations, use the following practices:

- Run vacuum operations on a regular schedule.

If you load your tables in small increments (such as daily updates that represent a small percentage of the total number of rows in the table), running VACUUM regularly will help ensure that individual vacuum operations go quickly.

- Run the largest load first.

If you need to load a new table with multiple COPY operations, run the largest load first. When you run an initial load into a new or truncated table, all of the data is loaded directly into the sorted region, so no vacuum is required.

- Truncate a table instead of deleting all of the rows.

Deleting rows from a table does not reclaim the space that the rows occupied until you perform a vacuum operation; however, truncating a table empties the table and reclaims the disk space, so no vacuum is required. Alternatively, drop the table and re-create it.

- Truncate or drop test tables.

If you are loading a small number of rows into a table for test purposes, don't delete the rows when you are done. Instead, truncate the table and reload those rows as part of the subsequent production load operation.

- Perform a deep copy.

If a table that uses a compound sort key table has a large unsorted region, a deep copy is much faster than a vacuum. A deep copy recreates and repopulates a table by using a bulk insert, which automatically re-sorts the table. If a table has a large unsorted region, a deep copy is much faster than a vacuum. The trade off is that you cannot make concurrent updates during a deep copy operation, which you can do during a vacuum. For more information, see [Amazon Redshift best practices for designing queries \(p. 21\)](#).

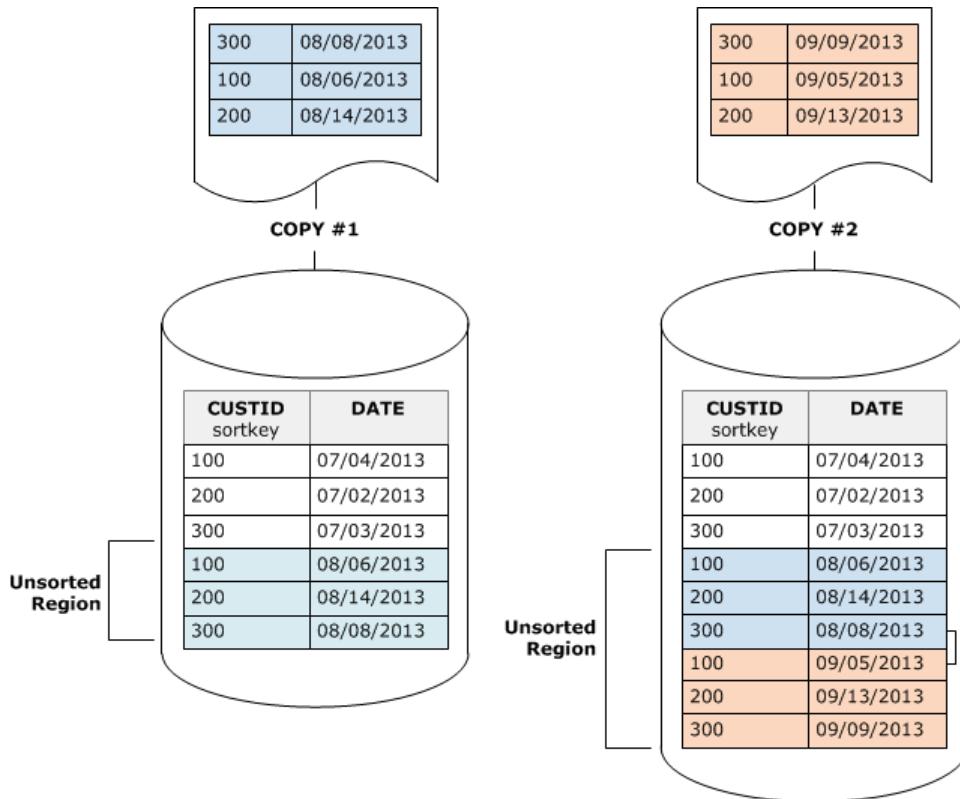
Managing the volume of merged rows

If a vacuum operation needs to merge new rows into a table's sorted region, the time required for a vacuum will increase as the table grows larger. You can improve vacuum performance by reducing the number of rows that must be merged.

Before a vacuum, a table consists of a sorted region at the head of the table, followed by an unsorted region, which grows whenever rows are added or updated. When a set of rows is added by a COPY

operation, the new set of rows is sorted on the sort key as it is added to the unsorted region at the end of the table. The new rows are ordered within their own set, but not within the unsorted region.

The following diagram illustrates the unsorted region after two successive COPY operations, where the sort key is CUSTID. For simplicity, this example shows a compound sort key, but the same principles apply to interleaved sort keys, except that the impact of the unsorted region is greater for interleaved tables.



A vacuum restores the table's sort order in two stages:

1. Sort the unsorted region into a newly-sorted region.

The first stage is relatively cheap, because only the unsorted region is rewritten. If the range of sort key values of the newly sorted region is higher than the existing range, only the new rows need to be rewritten, and the vacuum is complete. For example, if the sorted region contains ID values 1 to 500 and subsequent copy operations add key values greater than 500, then only the unsorted region needs to be rewritten.

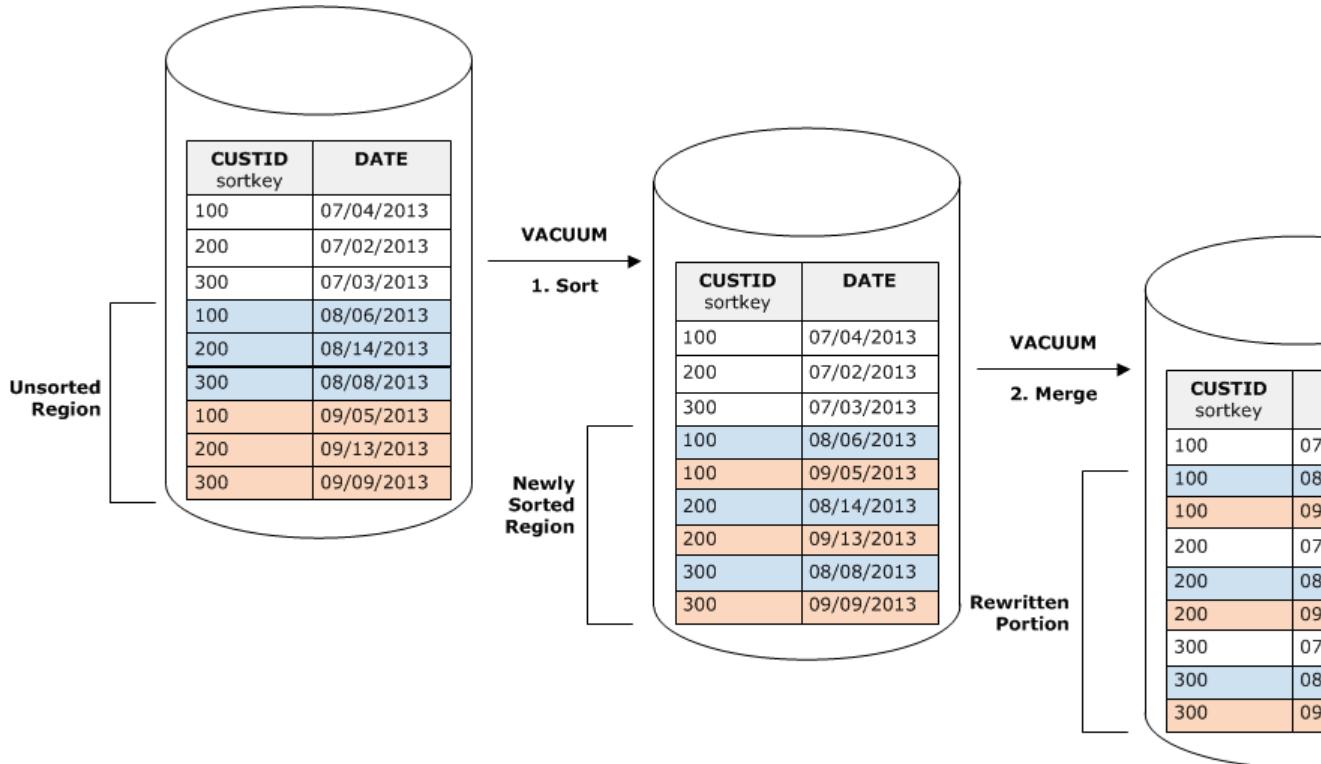
2. Merge the newly-sorted region with the previously-sorted region.

If the keys in the newly sorted region overlap the keys in the sorted region, then VACUUM needs to merge the rows. Starting at the beginning of the newly-sorted region (at the lowest sort key), the vacuum writes the merged rows from the previously sorted region and the newly sorted region into a new set of blocks.

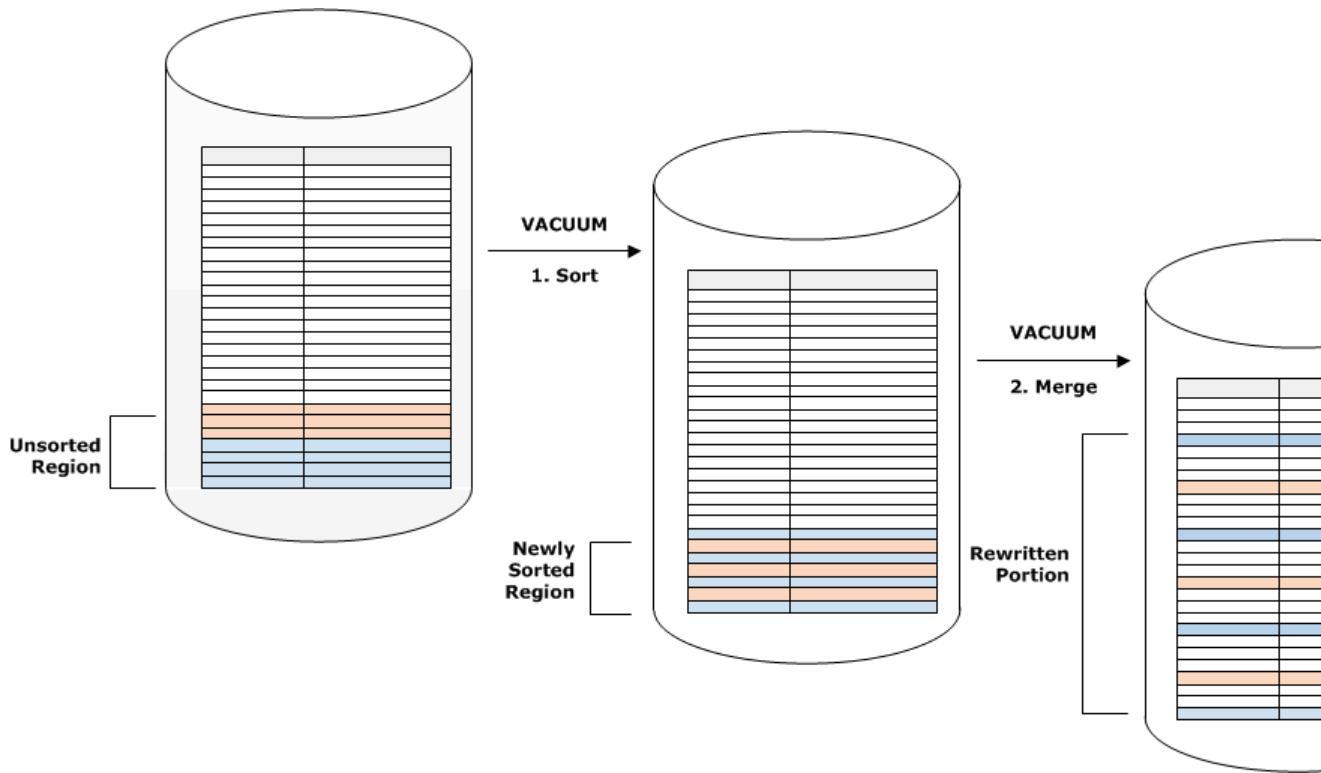
The extent to which the new sort key range overlaps the existing sort keys determines the extent to which the previously-sorted region will need to be rewritten. If the unsorted keys are scattered throughout the existing sort range, a vacuum might need to rewrite existing portions of the table.

The following diagram shows how a vacuum would sort and merge rows that are added to a table where CUSTID is the sort key. Because each copy operation adds a new set of rows with key values that overlap

the existing keys, almost the entire table needs to be rewritten. The diagram shows single sort and merge, but in practice, a large vacuum consists of a series of incremental sort and merge steps.



If the range of sort keys in a set of new rows overlaps the range of existing keys, the cost of the merge stage continues to grow in proportion to the table size as the table grows while the cost of the sort stage remains proportional to the size of the unsorted region. In such a case, the cost of the merge stage overshadows the cost of the sort stage, as the following diagram shows.



To determine what proportion of a table was remerged, query SVV_VACUUM_SUMMARY after the vacuum operation completes. The following query shows the effect of six successive vacuums as CUSTSALES grew larger over time.

select * from svv_vacuum_summary where table_name = 'custsales';								
table_name	xid	sort_max_merge_partitions	merge_increments	elapsed_time	row_delta	sortedrow_delta	block_delta	
custsales	7072	47	3	2	143918314	0	88297472	1524
custsales	7122	47	3	3	164157882	0	88297472	772
custsales	7212	47	3	4	187433171	0	88297472	767
custsales	7289	47	3	4	255482945	0	88297472	770
custsales	7420	47	3	5	316583833	0	88297472	769
custsales	9007	47	3	6	306685472	0	88297472	772
(6 rows)								

The merge_increments column gives an indication of the amount of data that was merged for each vacuum operation. If the number of merge increments over consecutive vacuums increases in proportion to the growth in table size, it indicates that each vacuum operation is remerging an increasing number of rows in the table because the existing and newly sorted regions overlap.

Loading your data in sort key order

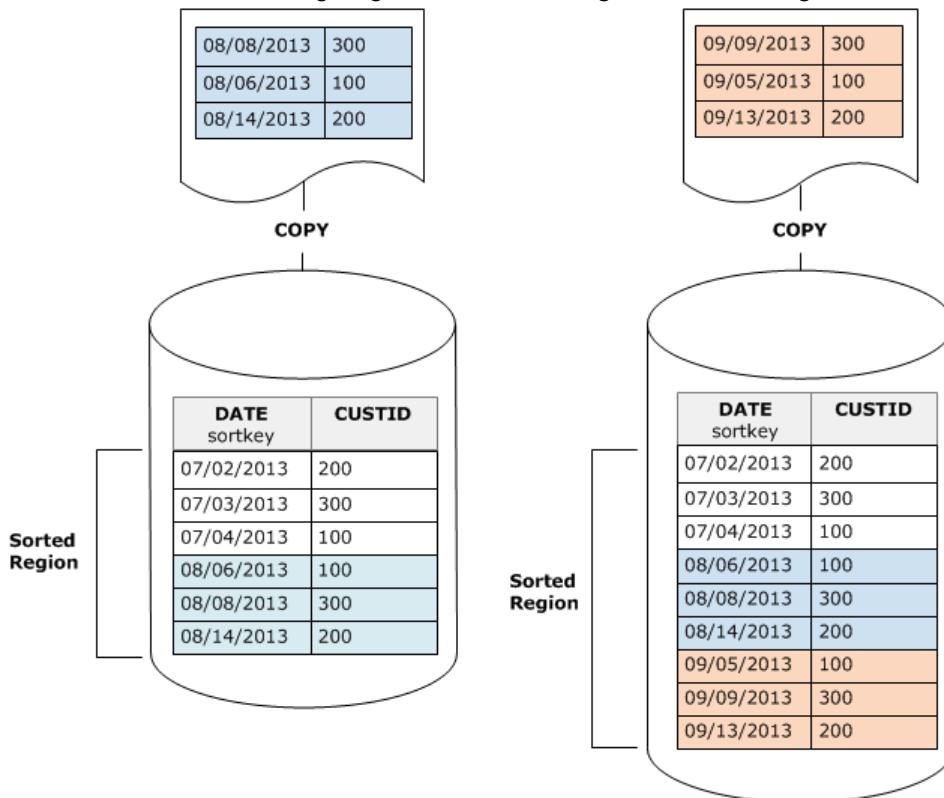
If you load your data in sort key order using a COPY command, you might reduce or even remove the need to vacuum.

COPY automatically adds new rows to the table's sorted region when all of the following are true:

- The table uses a compound sort key with only one sort column.
- The sort column is NOT NULL.
- The table is 100 percent sorted or empty.
- All the new rows are higher in sort order than the existing rows, including rows marked for deletion. In this instance, Amazon Redshift uses the first eight bytes of the sort key to determine sort order.

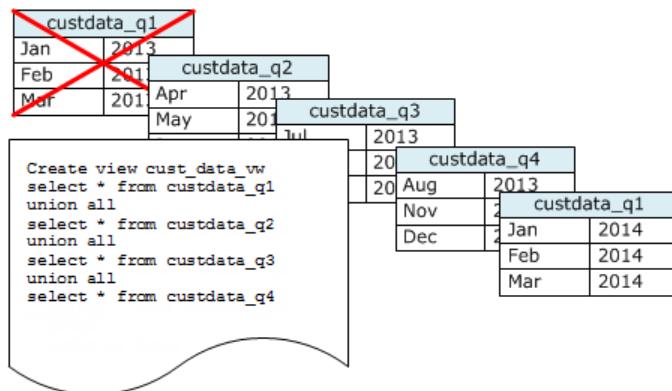
For example, suppose you have a table that records customer events using a customer ID and time. If you sort on customer ID, it's likely that the sort key range of new rows added by incremental loads will overlap the existing range, as shown in the previous example, leading to an expensive vacuum operation.

If you set your sort key to a timestamp column, your new rows will be appended in sort order at the end of the table, as the following diagram shows, reducing or even removing the need to vacuum.



Using time series tables

If you maintain data for a rolling time period, use a series of tables, as the following diagram illustrates.



Create a new table each time you add a set of data, then delete the oldest table in the series. You gain a double benefit:

- You avoid the added cost of deleting rows, because a `DROP TABLE` operation is much more efficient than a mass `DELETE`.
- If the tables are sorted by timestamp, no vacuum is needed. If each table contains data for one month, a vacuum will at most have to rewrite one month's worth of data, even if the tables are not sorted by timestamp.

You can create a `UNION ALL` view for use by reporting queries that hides the fact that the data is stored in multiple tables. If a query filters on the sort key, the query planner can efficiently skip all the tables that aren't used. A `UNION ALL` can be less efficient for other types of queries, so you should evaluate query performance in the context of all queries that use the tables.

Managing concurrent write operations

Topics

- [Serializable isolation \(p. 119\)](#)
- [Write and read/write operations \(p. 122\)](#)
- [Concurrent write examples \(p. 123\)](#)

Amazon Redshift allows tables to be read while they are being incrementally loaded or modified.

In some traditional data warehousing and business intelligence applications, the database is available to users only when the nightly load is complete. In such cases, no updates are allowed during regular work hours, when analytic queries are run and reports are generated; however, an increasing number of applications remain live for long periods of the day or even all day, making the notion of a load window obsolete.

Amazon Redshift supports these types of applications by allowing tables to be read while they are being incrementally loaded or modified. Queries simply see the latest committed version, or *snapshot*, of the data, rather than waiting for the next version to be committed. If you want a particular query to wait for a commit from another write operation, you have to schedule it accordingly.

The following topics describe some of the key concepts and use cases that involve transactions, database snapshots, updates, and concurrent behavior.

Serializable isolation

Some applications require not only concurrent querying and loading, but also the ability to write to multiple tables or the same table concurrently. In this context, *concurrently* means overlapping, not scheduled to run at precisely the same time. Two transactions are considered to be concurrent if the second one starts before the first commits. Concurrent operations can originate from different sessions that are controlled either by the same user or by different users.

Note

Amazon Redshift supports a default *automatic commit* behavior in which each separately run SQL command commits individually. If you enclose a set of commands in a transaction block (defined by [BEGIN \(p. 669\)](#) and [END \(p. 879\)](#) statements), the block commits as one transaction, so you can roll it back if necessary. Exceptions to this behavior are the TRUNCATE and VACUUM commands, which automatically commit all outstanding changes made in the current transaction.

Some SQL clients issue BEGIN and COMMIT commands automatically, so the client controls whether a group of statements are run as a transaction or each individual statement is run as its own transaction. Check the documentation for the interface you are using. For example, when using the Amazon Redshift JDBC driver, a JDBC PreparedStatement with a query string that contains multiple (semicolon separated) SQL commands runs all the statements as a single transaction. In contrast, if you use SQL Workbench/J and set AUTO COMMIT ON, then if you run multiple statements, each statement runs as its own transaction.

Concurrent write operations are supported in Amazon Redshift in a protective way, using write locks on tables and the principle of *serializable isolation*. Serializable isolation preserves the illusion that a transaction running against a table is the only transaction that is running against that table. For example, two concurrently running transactions, T1 and T2, must produce the same results as at least one of the following:

- T1 and T2 run serially in that order.
- T2 and T1 run serially in that order.

Concurrent transactions are invisible to each other; they cannot detect each other's changes. Each concurrent transaction will create a snapshot of the database at the beginning of the transaction. A database snapshot is created within a transaction on the first occurrence of most SELECT statements, DML commands such as COPY, DELETE, INSERT, UPDATE, and TRUNCATE, and the following DDL commands:

- ALTER TABLE (to add or drop columns)
- CREATE TABLE
- DROP TABLE
- TRUNCATE TABLE

If any serial execution of the concurrent transactions produces the same results as their concurrent execution, those transactions are deemed "serializable" and can be run safely. If no serial execution of those transactions can produce the same results, the transaction that runs a statement that might break the ability to serialize is stopped and rolled back.

System catalog tables (PG) and other Amazon Redshift system tables (STL and STV) are not locked in a transaction. Therefore, changes to database objects that arise from DDL and TRUNCATE operations are visible on commit to any concurrent transactions.

For example, suppose that table A exists in the database when two concurrent transactions, T1 and T2, start. Suppose that T2 returns a list of tables by selecting from the PG_TABLES catalog table. Then T1 drops table A and commits, and then T2 lists the tables again. Table A is now no longer listed. If T2 tries to query the dropped table, Amazon Redshift returns a "relation does not exist" error. The catalog query

that returns the list of tables to T2 or checks that table A exists isn't subject to the same isolation rules as operations performed on user tables.

Transactions for updates to these tables run in a read committed isolation mode. PG-prefix catalog tables don't support snapshot isolation.

Serializable isolation for system tables and catalog tables

A database snapshot is also created in a transaction for any SELECT query that references a user-created table or Amazon Redshift system table (STL or STV). SELECT queries that don't reference any table don't create a new transaction database snapshot. INSERT, DELETE, and UPDATE statements that operate solely on system catalog tables (PG) also don't create a new transaction database snapshot.

How to fix serializable isolation errors

ERROR:1023 DETAIL: Serializable isolation violation on a table in Redshift

When Amazon Redshift detects a serializable isolation error, you see an error message such as the following.

```
ERROR:1023 DETAIL: Serializable isolation violation on table in Redshift
```

To address a serializable isolation error, you can try the following methods:

- Retry the canceled transaction.

Amazon Redshift detected that a concurrent workload is not serializable. It suggests gaps in the application logic, which can usually be worked around by retrying the transaction that encountered the error. If the issue persists, try one of the other methods.

- Move any operations that don't have to be in the same atomic transaction outside of the transaction.

This method applies when individual operations inside two transactions cross-reference each other in a way that can affect the outcome of the other transaction. For example, the following two sessions each start a transaction.

```
Session1_Redshift=# begin;
```

```
Session2_Redshift=# begin;
```

The result of a SELECT statement in each transaction might be affected by an INSERT statement in the other. In other words, suppose that you run the following statements serially, in any order. In every case, the result is one of the SELECT statements returning one more row than if the transactions were run concurrently. There is no order in which the operations can run serially that produces the same result as when run concurrently. Thus, the last operation that is run results in a serializable isolation error.

```
Session1_Redshift=# select * from tab1;
Session1_Redshift=# insert into tab2 values (1);
```

```
Session2_Redshift=# insert into tab1 values (1);
Session2_Redshift=# select * from tab2;
```

In many cases, the result of the SELECT statements isn't important. In other words, the atomicity of the operations in the transactions isn't important. In these cases, move the SELECT statements outside of their transactions, as shown in the following examples.

```
Session1_Redshift=# begin;
Session1_Redshift=# insert into tab1 values (1)
Session1_Redshift=# end;
Session1_Redshift=# select * from tab2;
```

```
Session2_Redshift # select * from tab1;
Session2_Redshift=# begin;
Session2_Redshift=# insert into tab2 values (1)
Session2_Redshift=# end;
```

In these examples, there are no cross-references in the transactions. The two INSERT statements don't affect each other. In these examples, there is at least one order in which the transactions can run serially and produce the same result as if run concurrently. This means that the transactions are serializable.

- Force serialization by locking all tables in each session.

The [LOCK \(p. 908\)](#) command blocks operations that can result in serializable isolation errors. When you use the LOCK command, be sure to do the following:

- Lock all tables affected by the transaction, including those affected by read-only SELECT statements inside the transaction.
- Lock tables in the same order, regardless of the order that operations are performed in.
- Lock all tables at the beginning of the transaction, before performing any operations.

ERROR:1018 DETAIL: Relation does not exist

When you run concurrent Amazon Redshift operations in different sessions, you see an error message such as the following.

```
ERROR: 1018 DETAIL: Relation does not exist.
```

Transactions in Amazon Redshift follow snapshot isolation. After a transaction begins, Amazon Redshift takes a snapshot of the database. For the entire lifecycle of the transaction, the transaction operates on the state of the database as reflected in the snapshot. If the transaction reads from a table that doesn't exist in the snapshot, it throws the 1018 error message shown previously. Even when another concurrent transaction creates a table after the transaction has taken the snapshot, the transaction can't read from the newly created table.

To address this serialization isolation error, you can try to move the start of the transaction to a point where you know the table exists.

If the table is created by another transaction, this point is at least after that transaction has been committed. Also, ensure that no concurrent transaction has been committed that might have dropped the table.

```
session1 = # BEGIN;
session1 = # DROP TABLE A;
session1 = # COMMIT;
```

```
session2 = # BEGIN;
```

```
session3 = # BEGIN;
session3 = # CREATE TABLE A (id INT);
```

```
session3 = # COMMIT;
```

```
session2 = # SELECT * FROM A;
```

The last operation that is run as the read operation by session2 results in a serializable isolation error. This error happens when session2 takes a snapshot and the table has already been dropped by a committed session1. In other words, even though a concurrent session3 has created the table, session2 doesn't see the table because it's not in the snapshot.

To resolve this error, you can reorder the sessions as follows.

```
session1 = # BEGIN;
session1 = # DROP TABLE A;
session1 = # COMMIT;
```

```
session3 = # BEGIN;
session3 = # CREATE TABLE A (id INT);
session3 = # COMMIT;
```

```
session2 = # BEGIN;
session2 = # SELECT * FROM A;
```

Now when session2 takes its snapshot, session3 has already been committed, and the table is in the database. Session2 can read from the table without any error.

Write and read/write operations

You can manage the specific behavior of concurrent write operations by deciding when and how to run different types of commands. The following commands are relevant to this discussion:

- COPY commands, which perform loads (initial or incremental)
- INSERT commands that append one or more rows at a time
- UPDATE commands, which modify existing rows
- DELETE commands, which remove rows

COPY and INSERT operations are pure write operations, but DELETE and UPDATE operations are read/write operations. (For rows to be deleted or updated, they have to be read first.) The results of concurrent write operations depend on the specific commands that are being run concurrently. COPY and INSERT operations against the same table are held in a wait state until the lock is released, then they proceed as normal.

UPDATE and DELETE operations behave differently because they rely on an initial table read before they do any writes. Given that concurrent transactions are invisible to each other, both UPDATEs and DELETEs have to read a snapshot of the data from the last commit. When the first UPDATE or DELETE releases its lock, the second UPDATE or DELETE needs to determine whether the data that it is going to work with is potentially stale. It will not be stale, because the second transaction does not obtain its snapshot of data until after the first transaction has released its lock.

Potential deadlock situation for concurrent write transactions

Whenever transactions involve updates of more than one table, there is always the possibility of concurrently running transactions becoming deadlocked when they both try to write to the same set of tables. A transaction releases all of its table locks at once when it either commits or rolls back; it does not relinquish locks one at a time.

For example, suppose that transactions T1 and T2 start at roughly the same time. If T1 starts writing to table A and T2 starts writing to table B, both transactions can proceed without conflict; however, if T1 finishes writing to table A and needs to start writing to table B, it will not be able to proceed because T2 still holds the lock on B. Conversely, if T2 finishes writing to table B and needs to start writing to table A, it will not be able to proceed either because T1 still holds the lock on A. Because neither transaction can release its locks until all its write operations are committed, neither transaction can proceed.

In order to avoid this kind of deadlock, you need to schedule concurrent write operations carefully. For example, you should always update tables in the same order in transactions and, if specifying locks, lock tables in the same order before you perform any DML operations.

Concurrent write examples

The following pseudo-code examples demonstrate how transactions either proceed or wait when they are run concurrently.

Concurrent COPY operations into the same table

Transaction 1 copies rows into the LISTING table:

```
begin;
copy listing from ...;
end;
```

Transaction 2 starts concurrently in a separate session and attempts to copy more rows into the LISTING table. Transaction 2 must wait until transaction 1 releases the write lock on the LISTING table, then it can proceed.

```
begin;
[waits]
copy listing from ;
end;
```

The same behavior would occur if one or both transactions contained an INSERT command instead of a COPY command.

Concurrent DELETE operations from the same table

Transaction 1 deletes rows from a table:

```
begin;
delete from listing where ...;
end;
```

Transaction 2 starts concurrently and attempts to delete rows from the same table. It will succeed because it waits for transaction 1 to complete before attempting to delete rows.

```
begin
[waits]
delete from listing where ;
end;
```

The same behavior would occur if one or both transactions contained an UPDATE command to the same table instead of a DELETE command.

Concurrent transactions with a mixture of read and write operations

In this example, transaction 1 deletes rows from the USERS table, reloads the table, runs a COUNT(*) query, and then ANALYZE, before committing:

```
begin;
delete one row from USERS table;
copy ;
select count(*) from users;
analyze ;
end;
```

Meanwhile, transaction 2 starts. This transaction attempts to copy additional rows into the USERS table, analyze the table, and then run the same COUNT(*) query as the first transaction:

```
begin;
[waits]
copy users from ...;
select count(*) from users;
analyze;
end;
```

The second transaction will succeed because it must wait for the first to complete. Its COUNT query will return the count based on the load it has completed.

Tutorial: Loading data from Amazon S3

In this tutorial, you walk through the process of loading data into your Amazon Redshift database tables from data files in an Amazon S3 bucket from beginning to end.

In this tutorial, you do the following:

- Download data files that use comma-separated value (CSV), character-delimited, and fixed width formats.
- Create an Amazon S3 bucket and then upload the data files to the bucket.
- Launch an Amazon Redshift cluster and create database tables.
- Use COPY commands to load the tables from the data files on Amazon S3.
- Troubleshoot load errors and modify your COPY commands to correct the errors.

Estimated time: 60 minutes

Estimated cost: \$1.00 per hour for the cluster

Prerequisites

You need the following prerequisites:

- An AWS account to launch an Amazon Redshift cluster and to create a bucket in Amazon S3.
- Your AWS credentials (IAM role) to load test data from Amazon S3. If you need a new IAM role, go to [Creating IAM roles](#).

- An SQL client such as the Amazon Redshift console query editor.

This tutorial is designed so that it can be taken by itself. In addition to this tutorial, we recommend completing the following tutorials to gain a more complete understanding of how to design and use Amazon Redshift databases:

- [Amazon Redshift Getting Started Guide](#) walks you through the process of creating an Amazon Redshift cluster and loading sample data.

Overview

You can add data to your Amazon Redshift tables either by using an INSERT command or by using a COPY command. At the scale and speed of an Amazon Redshift data warehouse, the COPY command is many times faster and more efficient than INSERT commands.

The COPY command uses the Amazon Redshift massively parallel processing (MPP) architecture to read and load data in parallel from multiple data sources. You can load from data files on Amazon S3, Amazon EMR, or any remote host accessible through a Secure Shell (SSH) connection. Or you can load directly from an Amazon DynamoDB table.

In this tutorial, you use the COPY command to load data from Amazon S3. Many of the principles presented here apply to loading from other data sources as well.

To learn more about using the COPY command, see these resources:

- [Amazon Redshift best practices for loading data \(p. 17\)](#)
- [Loading data from Amazon EMR \(p. 74\)](#)
- [Loading data from remote hosts \(p. 78\)](#)
- [Loading data from an Amazon DynamoDB table \(p. 84\)](#)

Steps

- [Step 1: Create a cluster \(p. 125\)](#)
- [Step 2: Download the data files \(p. 126\)](#)
- [Step 3: Upload the files to an Amazon S3 bucket \(p. 127\)](#)
- [Step 4: Create the sample tables \(p. 128\)](#)
- [Step 5: Run the COPY commands \(p. 130\)](#)
- [Step 6: Vacuum and analyze the database \(p. 142\)](#)
- [Step 7: Clean up your resources \(p. 142\)](#)

Step 1: Create a cluster

If you already have a cluster that you want to use, you can skip this step.

For the exercises in this tutorial, use a four-node cluster.

To create a cluster

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.

Using the navigation menu, choose the **Provisioned clusters dashboard**.

Important

Make sure that you have the necessary permissions to perform the cluster operations. For information on granting the necessary permissions, see [Authorizing Amazon Redshift to access AWS services](#).

2. At top right, choose the AWS Region in which you want to create the cluster. For the purposes of this tutorial, choose **US West (Oregon)**.
3. On the navigation menu, choose **Clusters**, then choose **Create cluster**. The **Create cluster** page appears.
4. On the **Create cluster** page enter parameters for your cluster. Choose your own values for the parameters, except change the following values:
 - Choose **dc2.large** for the node type.
 - Choose **4** for the **Number of nodes**.
 - In the **Cluster permissions** section, choose an IAM role from **Available IAM roles**. This role should be one that you previously created and that has access to Amazon S3. Then choose **Associate IAM role** to add it to the list of **Associated IAM roles** for the cluster.
5. Choose **Create cluster**.

Follow the [Amazon Redshift Getting Started Guide](#) steps to connect to your cluster from a SQL client and test a connection. You don't need to complete the remaining Getting Started steps to create tables, upload data, and try example queries.

Next step

[Step 2: Download the data files \(p. 126\)](#)

Step 2: Download the data files

In this step, you download a set of sample data files to your computer. In the next step, you upload the files to an Amazon S3 bucket.

To download the data files

1. Download the zipped file: [LoadingDataSampleFiles.zip](#).
2. Extract the files to a folder on your computer.
3. Verify that your folder contains the following files.

```
customer-fw-manifest
customer-fw.tbl-000
customer-fw.tbl-000.bak
customer-fw.tbl-001
customer-fw.tbl-002
customer-fw.tbl-003
customer-fw.tbl-004
customer-fw.tbl-005
customer-fw.tbl-006
customer-fw.tbl-007
customer-fw.tbl.log
dwdate-tab.tbl-000
dwdate-tab.tbl-001
dwdate-tab.tbl-002
dwdate-tab.tbl-003
dwdate-tab.tbl-004
dwdate-tab.tbl-005
```

```
dwdate-tab.tbl-006
dwdate-tab.tbl-007
part-csv.tbl-000
part-csv.tbl-001
part-csv.tbl-002
part-csv.tbl-003
part-csv.tbl-004
part-csv.tbl-005
part-csv.tbl-006
part-csv.tbl-007
```

Next step

[Step 3: Upload the files to an Amazon S3 bucket \(p. 127\)](#)

Step 3: Upload the files to an Amazon S3 bucket

In this step, you create an Amazon S3 bucket and upload the data files to the bucket.

To upload the files to an Amazon S3 bucket

1. Create a bucket in Amazon S3.

- a. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
- b. Click **Create Bucket**.
- c. In the **Bucket Name** box of the **Create a Bucket** dialog box, type a bucket name.

The bucket name you choose must be unique among all existing bucket names in Amazon S3. One way to help ensure uniqueness is to prefix your bucket names with the name of your organization. Bucket names must comply with certain rules. For more information, go to [Bucket restrictions and limitations](#) in the *Amazon Simple Storage Service User Guide*.

- d. Select a Region.

Create the bucket in the same Region as your cluster. If your cluster is in the US West (Oregon) Region, choose **US West (Oregon) Region (us-west-2)**.

- e. Choose **Create**.

When Amazon S3 successfully creates your bucket, the console displays your empty bucket in the **Buckets** panel.

2. Create a folder.

- a. Choose the name of the new bucket.
- b. Choose the **Actions** button, and click **Create Folder** in the drop-down list.
- c. Name the new folder **load**.

Note

The bucket that you created is not in a sandbox. In this exercise, you add objects to a real bucket. You're charged a nominal amount for the time that you store the objects in the bucket. For more information about Amazon S3 pricing, go to the [Amazon S3 pricing](#) page.

3. Upload the data files to the new Amazon S3 bucket.

- a. Choose the name of the data folder.
- b. In the Upload - Select Files wizard, choose **Add Files**.

Follow the Amazon S3 console instructions to upload all of the files you downloaded and extracted,

- c. Choose **Start Upload**.

User Credentials

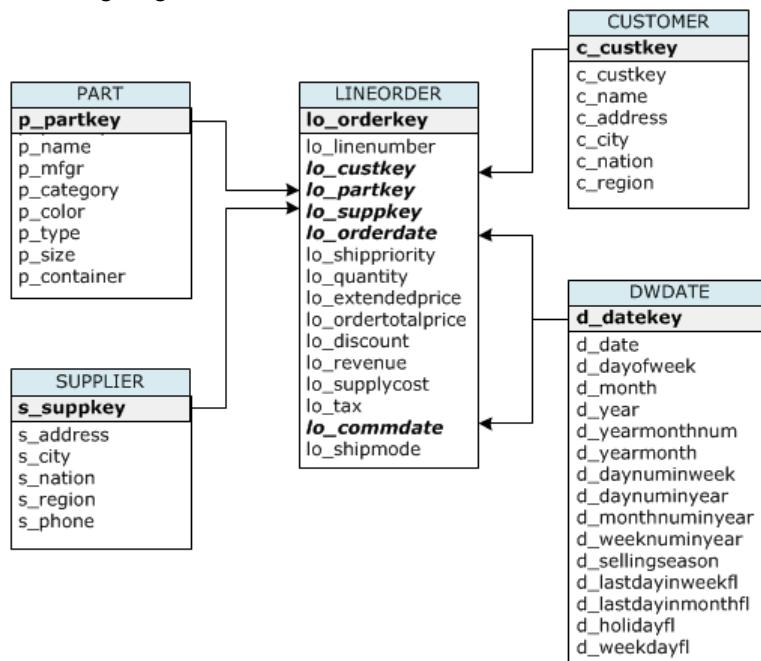
The Amazon Redshift COPY command must have access to read the file objects in the Amazon S3 bucket. If you use the same user credentials to create the Amazon S3 bucket and to run the Amazon Redshift COPY command, the COPY command has all necessary permissions. If you want to use different user credentials, you can grant access by using the Amazon S3 access controls. The Amazon Redshift COPY command requires at least ListBucket and GetObject permissions to access the file objects in the Amazon S3 bucket. For more information about controlling access to Amazon S3 resources, go to [Managing access permissions to your Amazon S3 resources](#).

Next step

[Step 4: Create the sample tables \(p. 128\)](#)

Step 4: Create the sample tables

For this tutorial, you use a set of five tables based on the Star Schema Benchmark (SSB) schema. The following diagram shows the SSB data model.



The SSB tables might already exist in the current database. If so, drop the tables to remove them from the database before you create them using the CREATE TABLE commands in the next step. The tables used in this tutorial might have different attributes than the existing tables.

To create the sample tables

1. To drop the SSB tables, run the following commands in your SQL client.

```
drop table part cascade;
```

```
drop table supplier;
drop table customer;
drop table dwdate;
drop table lineorder;
```

- Run the following CREATE TABLE commands in your SQL client.

```
CREATE TABLE part
(
    p_partkey      INTEGER NOT NULL,
    p_name         VARCHAR(22) NOT NULL,
    p_mfgr          VARCHAR(6),
    p_category      VARCHAR(7) NOT NULL,
    p_brand1        VARCHAR(9) NOT NULL,
    p_color          VARCHAR(11) NOT NULL,
    p_type           VARCHAR(25) NOT NULL,
    p_size            INTEGER NOT NULL,
    p_container      VARCHAR(10) NOT NULL
);

CREATE TABLE supplier
(
    s_suppkey      INTEGER NOT NULL,
    s_name          VARCHAR(25) NOT NULL,
    s_address        VARCHAR(25) NOT NULL,
    s_city           VARCHAR(10) NOT NULL,
    s_nation         VARCHAR(15) NOT NULL,
    s_region          VARCHAR(12) NOT NULL,
    s_phone           VARCHAR(15) NOT NULL
);

CREATE TABLE customer
(
    c_custkey      INTEGER NOT NULL,
    c_name          VARCHAR(25) NOT NULL,
    c_address        VARCHAR(25) NOT NULL,
    c_city           VARCHAR(10) NOT NULL,
    c_nation         VARCHAR(15) NOT NULL,
    c_region          VARCHAR(12) NOT NULL,
    c_phone           VARCHAR(15) NOT NULL,
    c_mktsegment     VARCHAR(10) NOT NULL
);

CREATE TABLE dwdate
(
    d_datekey      INTEGER NOT NULL,
    d_date          VARCHAR(19) NOT NULL,
    d_dayofweek      VARCHAR(10) NOT NULL,
    d_month          VARCHAR(10) NOT NULL,
    d_year            INTEGER NOT NULL,
    d_yeарmonthnum     INTEGER NOT NULL,
    d_yeарmonth      VARCHAR(8) NOT NULL,
    d_daynuminweek     INTEGER NOT NULL,
    d_daynuminmonth     INTEGER NOT NULL,
    d_daynuminyear     INTEGER NOT NULL,
    d_monthnuminyear     INTEGER NOT NULL,
    d_weeknuminyear     INTEGER NOT NULL,
    d_sellingseason      VARCHAR(13) NOT NULL,
    d_lastdayinweekfl     VARCHAR(1) NOT NULL,
    d_lastdayinmonthfl     VARCHAR(1) NOT NULL,
    d_holidayfl        VARCHAR(1) NOT NULL,
    d_weekdayfl        VARCHAR(1) NOT NULL
);

CREATE TABLE lineorder
(
```

```
    lo_orderkey      INTEGER NOT NULL,  
    lo_linenumber   INTEGER NOT NULL,  
    lo_custkey      INTEGER NOT NULL,  
    lo_partkey      INTEGER NOT NULL,  
    lo_suppkey      INTEGER NOT NULL,  
    lo_orderdate    INTEGER NOT NULL,  
    lo_orderpriority VARCHAR(15) NOT NULL,  
    lo_shippriority VARCHAR(1) NOT NULL,  
    lo_quantity     INTEGER NOT NULL,  
    lo_extendedprice INTEGER NOT NULL,  
    lo_ordertotalprice INTEGER NOT NULL,  
    lo_discount     INTEGER NOT NULL,  
    lo_revenue      INTEGER NOT NULL,  
    lo_supplycost   INTEGER NOT NULL,  
    lo_tax          INTEGER NOT NULL,  
    lo_commitdate   INTEGER NOT NULL,  
    lo_shipmode     VARCHAR(10) NOT NULL  
);
```

Next step

[Step 5: Run the COPY commands \(p. 130\)](#)

Step 5: Run the COPY commands

You run COPY commands to load each of the tables in the SSB schema. The COPY command examples demonstrate loading from different file formats, using several COPY command options, and troubleshooting load errors.

Topics

- [COPY command syntax \(p. 130\)](#)
- [Loading the SSB tables \(p. 132\)](#)

COPY command syntax

The basic [COPY \(p. 677\)](#) command syntax is as follows.

```
COPY table_name [ column_list ] FROM data_source CREDENTIALS access_credentials [options]
```

To run a COPY command, you provide the following values.

Table name

The target table for the COPY command. The table must already exist in the database. The table can be temporary or persistent. The COPY command appends the new input data to any existing rows in the table.

Column list

By default, COPY loads fields from the source data to the table columns in order. You can optionally specify a *column list*, that is a comma-separated list of column names, to map data fields to specific columns. You don't use column lists in this tutorial. For more information, see [Column List \(p. 696\)](#) in the COPY command reference.

Data source

You can use the COPY command to load data from an Amazon S3 bucket, an Amazon EMR cluster, a remote host using an SSH connection, or an Amazon DynamoDB table. For this tutorial, you load from data files in an Amazon S3 bucket. When loading from Amazon S3, you must provide the name of the bucket and the location of the data files. To do this, provide either an object path for the data files or the location of a manifest file that explicitly lists each data file and its location.

- Key prefix

An object stored in Amazon S3 is uniquely identified by an object key, which includes the bucket name, folder names, if any, and the object name. A *key prefix* refers to a set of objects with the same prefix. The object path is a key prefix that the COPY command uses to load all objects that share the key prefix. For example, the key prefix `custdata.txt` can refer to a single file or to a set of files, including `custdata.txt.001`, `custdata.txt.002`, and so on.

- Manifest file

In some cases, you might need to load files with different prefixes, for example from multiple buckets or folders. In others, you might need to exclude files that share a prefix. In these cases, you can use a manifest file. A *manifest file* explicitly lists each load file and its unique object key. You use a manifest file to load the PART table later in this tutorial.

Credentials

To access the AWS resources that contain the data to load, you must provide AWS access credentials for a user with sufficient privileges. These credentials include an IAM role Amazon Resource Name (ARN).

To load data from Amazon S3, the credentials must include `ListBucket` and `GetObject` permissions.

Additional credentials are required if your data is encrypted. For more information, see [Authorization parameters \(p. 693\)](#) in the COPY command reference. For more information about managing access, go to [Managing access permissions to your Amazon S3 resources](#).

Options

You can specify a number of parameters with the COPY command to specify file formats, manage data formats, manage errors, and control other features. In this tutorial, you use the following COPY command options and features:

- Key prefix

For information on how to load from multiple files by specifying a key prefix, see [Load the PART table using NULL AS \(p. 133\)](#).

- CSV format

For information on how to load data that is in CSV format, see [Load the PART table using NULL AS \(p. 133\)](#).

- NULL AS

For information on how to load PART using the NULL AS option, see [Load the PART table using NULL AS \(p. 133\)](#).

- Character-delimited format

For information on how to use the DELIMITER option, see [Load the SUPPLIER table using REGION \(p. 135\)](#).

- REGION

For information on how to use the REGION option, see [Load the SUPPLIER table using REGION \(p. 135\)](#).

- Fixed-format width

For information on how to load the CUSTOMER table from fixed-width data, see [Load the CUSTOMER table using MANIFEST \(p. 136\)](#).

- MAXERROR

For information on how to use the MAXERROR option, see [Load the CUSTOMER table using MANIFEST \(p. 136\)](#).

- ACCEPTINVCHARS

For information on how to use the ACCEPTINVCHARS option, see [Load the CUSTOMER table using MANIFEST \(p. 136\)](#).

- MANIFEST

For information on how to use the MANIFEST option, see [Load the CUSTOMER table using MANIFEST \(p. 136\)](#).

- DATEFORMAT

For information on how to use the DATEFORMAT option, see [Load the DWDATETABLE using DATEFORMAT \(p. 139\)](#).

- GZIP, LZOP and BZIP2

For information on how to compress your files, see [Load the LINEORDER table using multiple files \(p. 140\)](#).

- COMPUPDATE

For information on how to use the COMPUPDATE option, see [Load the LINEORDER table using multiple files \(p. 140\)](#).

- Multiple files

For information on how to load multiple files, see [Load the LINEORDER table using multiple files \(p. 140\)](#).

Loading the SSB tables

You use the following COPY commands to load each of the tables in the SSB schema. The command to each table demonstrates different COPY options and troubleshooting techniques.

To load the SSB tables, follow these steps:

1. [Replace the bucket name and AWS credentials \(p. 132\)](#)
2. [Load the PART table using NULL AS \(p. 133\)](#)
3. [Load the SUPPLIER table using REGION \(p. 135\)](#)
4. [Load the CUSTOMER table using MANIFEST \(p. 136\)](#)
5. [Load the DWDATETABLE using DATEFORMAT \(p. 139\)](#)
6. [Load the LINEORDER table using multiple files \(p. 140\)](#)

Replace the bucket name and AWS credentials

The COPY commands in this tutorial are presented in the following format.

```
copy table from 's3://<your-bucket-name>/load/key_prefix'  
credentials 'aws_iam_role=arn:aws:iam:<aws-account-id>:role/<role-name>'  
options;
```

For each COPY command, do the following:

1. Replace `<your-bucket-name>` with the name of a bucket in the same region as your cluster.

This step assumes the bucket and the cluster are in the same region. Alternatively, you can specify the region using the [REGION \(p. 686\)](#) option with the COPY command.

2. Replace `<aws-account-id>` and `<role-name>` with your own AWS account and IAM role. The segment of the credentials string that is enclosed in single quotation marks must not contain any spaces or line breaks.

Load the PART table using NULL AS

In this step, you use the CSV and NULL AS options to load the PART table.

The COPY command can load data from multiple files in parallel, which is much faster than loading from a single file. To demonstrate this principle, the data for each table in this tutorial is split into eight files, even though the files are very small. In a later step, you compare the time difference between loading from a single file and loading from multiple files. For more information, see [Loading data files \(p. 18\)](#).

Key prefix

You can load from multiple files by specifying a key prefix for the file set, or by explicitly listing the files in a manifest file. In this step, you use a key prefix. In a later step, you use a manifest file. The key prefix '`s3://mybucket/load/part-csv.tbl`' loads the following set of the files in the load folder.

```
part-csv.tbl-000
part-csv.tbl-001
part-csv.tbl-002
part-csv.tbl-003
part-csv.tbl-004
part-csv.tbl-005
part-csv.tbl-006
part-csv.tbl-007
```

CSV format

CSV, which stands for comma separated values, is a common format used for importing and exporting spreadsheet data. CSV is more flexible than comma-delimited format because it enables you to include quoted strings within fields. The default quotation mark character for COPY from CSV format is a double quotation mark (" "), but you can specify another quotation mark character by using the QUOTE AS option. When you use the quotation mark character within the field, escape the character with an additional quotation mark character.

The following excerpt from a CSV-formatted data file for the PART table shows strings enclosed in double quotation marks ("LARGE ANODIZED BRASS"). It also shows a string enclosed in two double quotation marks within a quoted string ("MEDIUM ""BURNISHED"" TIN").

```
15,dark sky,MFGR#3,MFGR#47,MFGR#3438,indigo,"LARGE ANODIZED BRASS",45,LG CASE
22,floral beige,MFGR#4,MFGR#44,MFGR#4421,medium,"PROMO, POLISHED BRASS",19,LG DRUM
23,bisque slate,MFGR#4,MFGR#41,MFGR#4137,firebrick,"MEDIUM ""BURNISHED"" TIN",42,JUMBO JAR
```

The data for the PART table contains characters that cause COPY to fail. In this exercise, you troubleshoot the errors and correct them.

To load data that is in CSV format, add `csv` to your COPY command. Run the following command to load the PART table.

```
copy part from 's3://<your-bucket-name>/load/part-csv.tbl'
```

```
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'  
csv;
```

You might get an error message similar to the following.

```
An error occurred when executing the SQL command:  
copy part from 's3://mybucket/load/part-csv.tbl'  
credentials' ...  
  
ERROR: Load into table 'part' failed. Check 'stl_load_errors' system table for details.  
[SQL State=XX000]  
  
Execution time: 1.46s  
  
1 statement(s) failed.  
1 statement(s) failed.
```

To get more information about the error, query the `STL_LOAD_ERRORS` table. The following query uses the `SUBSTRING` function to shorten columns for readability and uses `LIMIT 10` to reduce the number of rows returned. You can adjust the values in `substring(filename, 22, 25)` to allow for the length of your bucket name.

```
select query, substring(filename,22,25) as filename, line_number as line,  
substring(colname,0,12) as column, type, position as pos, substring(raw_line,0,30) as  
line_text,  
substring(raw_field_value,0,15) as field_text,  
substring(err_reason,0,45) as reason  
from stl_load_errors  
order by query desc  
limit 10;
```

query	filename	line	column	type	pos
333765	part-csv.tbl-000	1			0
line_text	field_text			reason	
15,NUL next,				Missing newline: Unexpected character 0x2c f	

NULL AS

The `part-csv.tbl` data files use the NUL terminator character (`\x000` or `\x0`) to indicate NULL values.

Note

Despite very similar spelling, NUL and NULL are not the same. NUL is a UTF-8 character with codepoint `x000` that is often used to indicate end of record (EOR). NULL is a SQL value that represents an absence of data.

By default, `COPY` treats a NUL terminator character as an EOR character and terminates the record, which often results in unexpected results or an error. There is no single standard method of indicating NULL in text data. Thus, the `NULL AS` `COPY` command option enables you to specify which character to substitute with NULL when loading the table. In this example, you want `COPY` to treat the NUL terminator character as a NULL value.

Note

The table column that receives the NULL value must be configured as *nullable*. That is, it must not include the NOT NULL constraint in the `CREATE TABLE` specification.

To load PART using the `NULL AS` option, run the following `COPY` command.

```
copy part from 's3://<your-bucket-name>/load/part-csv.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
csv
null as '\000';
```

To verify that COPY loaded NULL values, run the following command to select only the rows that contain NULL.

```
select p_partkey, p_name, p_mfgr, p_category from part where p_mfgr is null;
```

p_partkey	p_name	p_mfgr	p_category
15	NUL next		MFGR#47
81	NUL next		MFGR#23
133	NUL next		MFGR#44

(2 rows)

Load the SUPPLIER table using REGION

In this step, you use the DELIMITER and REGION options to load the SUPPLIER table.

Note

The files for loading the SUPPLIER table are provided in an AWS sample bucket. You don't need to upload files for this step.

Character-Delimited Format

The fields in a character-delimited file are separated by a specific character, such as a pipe character (|), a comma (,) or a tab (\t). Character-delimited files can use any single ASCII character, including one of the nonprinting ASCII characters, as the delimiter. You specify the delimiter character by using the DELIMITER option. The default delimiter is a pipe character (|).

The following excerpt from the data for the SUPPLIER table uses pipe-delimited format.

```
1|1|257368|465569|41365|19950218|2-HIGH|0|17|2608718|9783671|4|2504369|92072|2|19950331|
TRUCK
1|2|257368|201928|8146|19950218|2-HIGH|0|36|6587676|9783671|9|5994785|109794|6|19950416|
MAIL
```

REGION

Whenever possible, you should locate your load data in the same AWS region as your Amazon Redshift cluster. If your data and your cluster are in the same region, you reduce latency and avoid cross-region data transfer costs. For more information, see [Amazon Redshift best practices for loading data \(p. 17\)](#)

If you must load data from a different AWS region, use the REGION option to specify the AWS region in which the load data is located. If you specify a region, all of the load data, including manifest files, must be in the named region. For more information, see [REGION \(p. 686\)](#).

If your cluster is in the US East (N. Virginia) Region, run the following command to load the SUPPLIER table from pipe-delimited data in an Amazon S3 bucket located in the US West (Oregon) Region. For this example, do not change the bucket name.

```
copy supplier from 's3://awssampledbuswest2/ssbgz/supplier.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
delimiter '|'
gzip
```

```
region 'us-west-2';
```

If your cluster is *not* in the US East (N. Virginia) region, run the following command to load the SUPPLIER table from pipe-delimited data in an Amazon S3 bucket located in the US East (N. Virginia) region. For this example, do not change the bucket name.

```
copy supplier from 's3://awssampledbs/bzg/supplier.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
delimiter '|'
gzip
region 'us-east-1';
```

Load the CUSTOMER table using MANIFEST

In this step, you use the FIXEDWIDTH, MAXERROR, ACCEPTINVCHARS, and MANIFEST options to load the CUSTOMER table.

The sample data for this exercise contains characters that cause errors when COPY attempts to load them. You use the MAXERRORS option and the STL_LOAD_ERRORS system table to troubleshoot the load errors and then use the ACCEPTINVCHARS and MANIFEST options to eliminate the errors.

Fixed-Width Format

Fixed-width format defines each field as a fixed number of characters, rather than separating fields with a delimiter. The following excerpt from the data for the CUSTOMER table uses fixed-width format.

1	Customer#000000001	IVhzIApeRb	MOROCCO	OMOROCCO	AFRICA	25-705
2	Customer#000000002	XSTf4,NCwDVawNe6tE	JORDAN	6JORDAN	MIDDLE EAST	23-453
3	Customer#000000003	MG9kdTD	ARGENTINA	5ARGENTINAAMERICA	AMERICA	11-783

The order of the label/width pairs must match the order of the table columns exactly. For more information, see [FIXEDWIDTH \(p. 697\)](#).

The fixed-width specification string for the CUSTOMER table data is as follows.

```
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
c_region :12, c_phone:15,c_mktsegment:10'
```

To load the CUSTOMER table from fixed-width data, run the following command.

```
copy customer
from 's3://<your-bucket-name>/load/customer-fw.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15, c_region :12,
c_phone:15,c_mktsegment:10';
```

You should get an error message, similar to the following.

```
An error occurred when executing the SQL command:
copy customer
from 's3://mybucket/load/customer-fw.tbl'
credentials'...

ERROR: Load into table 'customer' failed. Check 'stl_load_errors' system table for
details. [SQL State=XX000]

Execution time: 2.95s
```

1 statement(s) failed.

MAXERROR

By default, the first time COPY encounters an error, the command fails and returns an error message. To save time during testing, you can use the MAXERROR option to instruct COPY to skip a specified number of errors before it fails. Because we expect errors the first time we test loading the CUSTOMER table data, add maxerror 10 to the COPY command.

To test using the FIXEDWIDTH and MAXERROR options, run the following command.

```
copy customer
from 's3://<your-bucket-name>/load/customer-fw.tbl'
credentials 'aws_iam_role=arn:aws:iam:<aws-account-id>:role/<role-name>'
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15, c_region :12,
c_phone:15,c_mktsegment:10'
maxerror 10;
```

This time, instead of an error message, you get a warning message similar to the following.

Warnings:
Load into table 'customer' completed, 112497 record(s) loaded successfully.
Load into table 'customer' completed, 7 record(s) could not be loaded. Check 'stl_load_errors' system table for details.

The warning indicates that COPY encountered seven errors. To check the errors, query the ST_L_LOAD_ERRORS table, as shown in the following example.

```
select query, substring(filename,22,25) as filename, line_number as line,
substring(colname,0,12) as column, type, position as pos, substring(raw_line,0,30) as
line_text,
substring(raw_field_value,0,15) as field_text,
substring(err_reason,0,45) as error_reason
from stl_load_errors
order by query desc, filename
limit 7;
```

The results of the ST_L_LOAD_ERRORS query should look similar to the following.

query	filename	line	column	type	pos	
line_text		field_text		error_reason		
334489	customer-fw.tbl.log	2	c_custkey	int4	-1	customer-fw.tbl
	customer-f	Invalid digit, Value 'c', Pos 0, Type: Integ				
334489	customer-fw.tbl.log	6	c_custkey	int4	-1	Complete
	Complete	Invalid digit, Value 'C', Pos 0, Type: Integ				
334489	customer-fw.tbl.log	3	c_custkey	int4	-1	#Total rows
	#Total row	Invalid digit, Value '#', Pos 0, Type: Integ				
334489	customer-fw.tbl.log	5	c_custkey	int4	-1	#Status
	#Status	Invalid digit, Value '#', Pos 0, Type: Integ				
334489	customer-fw.tbl.log	1	c_custkey	int4	-1	#Load file
	#Load file	Invalid digit, Value '#', Pos 0, Type: Integ				
334489	customer-fw.tbl000	1	c_address	varchar	34	1
	Customer#000000001	.Mayag.ezR	String contains invalid or unsupported UTF8			
334489	customer-fw.tbl000	1	c_address	varchar	34	1
	Customer#000000001	.Mayag.ezR	String contains invalid or unsupported UTF8			
(7 rows)						

By examining the results, you can see that there are two messages in the `error_reasons` column:

- Invalid digit, Value '#', Pos 0, Type: Integ

These errors are caused by the `customer-fw.tbl.log` file. The problem is that it is a log file, not a data file, and should not be loaded. You can use a manifest file to avoid loading the wrong file.

- String contains invalid or unsupported UTF8

The `VARCHAR` data type supports multibyte UTF-8 characters up to three bytes. If the load data contains unsupported or invalid characters, you can use the `ACCEPTINVCHARS` option to replace each invalid character with a specified alternative character.

Another problem with the load is more difficult to detect—the load produced unexpected results. To investigate this problem, run the following command to query the `CUSTOMER` table.

```
select c_custkey, c_name, c_address
from customer
order by c_custkey
limit 10;
```

c_custkey	c_name	c_address
2	Customer#000000002	XSTf4,NCwDVaWNe6tE
2	Customer#000000002	XSTf4,NCwDVaWNe6tE
3	Customer#000000003	MG9kdTD
3	Customer#000000003	MG9kdTD
4	Customer#000000004	XxVSJsL
4	Customer#000000004	XxVSJsL
5	Customer#000000005	KvpyuHCplrB84WgAi
5	Customer#000000005	KvpyuHCplrB84WgAi
6	Customer#000000006	sKZz0CsnMD7mp4Xd0YrBvx
6	Customer#000000006	sKZz0CsnMD7mp4Xd0YrBvx

The rows should be unique, but there are duplicates.

Another way to check for unexpected results is to verify the number of rows that were loaded. In our case, 100000 rows should have been loaded, but the load message reported loading 112497 records. The extra rows were loaded because the `COPY` loaded an extraneous file, `customer-fw.tbl10000.bak`.

In this exercise, you use a manifest file to avoid loading the wrong files.

ACCEPTINVCHARS

By default, when `COPY` encounters a character that is not supported by the column's data type, it skips the row and returns an error. For information about invalid UTF-8 characters, see [Multibyte character load errors \(p. 92\)](#).

You could use the `MAXERRORS` option to ignore errors and continue loading, then query `STL_LOAD_ERRORS` to locate the invalid characters, and then fix the data files. However, `MAXERRORS` is best used for troubleshooting load problems and should generally not be used in a production environment.

The `ACCEPTINVCHARS` option is usually a better choice for managing invalid characters. `ACCEPTINVCHARS` instructs `COPY` to replace each invalid character with a specified valid character and continue with the load operation. You can specify any valid ASCII character, except `NUL`, as the

replacement character. The default replacement character is a question mark (?). COPY replaces multibyte characters with a replacement string of equal length. For example, a 4-byte character would be replaced with '????'.

COPY returns the number of rows that contained invalid UTF-8 characters. It also adds an entry to the STL_REPLACEMENTS system table for each affected row, up to a maximum of 100 rows per node slice. Additional invalid UTF-8 characters are also replaced, but those replacement events are not recorded.

ACCEPTINVCHARS is valid only for VARCHAR columns.

For this step, you add the ACCEPTINVCHARS with the replacement character '^'.

MANIFEST

When you COPY from Amazon S3 using a key prefix, there is a risk that you might load unwanted tables. For example, the 's3://mybucket/load/' folder contains eight data files that share the key prefix customer-fw.tbl: customer-fw.tbl0000, customer-fw.tbl0001, and so on. However, the same folder also contains the extraneous files customer-fw.tbl.log and customer-fw.tbl-0001.bak.

To ensure that you load all of the correct files, and only the correct files, use a manifest file. The manifest is a text file in JSON format that explicitly lists the unique object key for each source file to be loaded. The file objects can be in different folders or different buckets, but they must be in the same region. For more information, see [MANIFEST \(p. 686\)](#).

The following shows the customer-fw-manifest text.

```
{  
  "entries": [  
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-000"},  
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-001"},  
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-002"},  
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-003"},  
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-004"},  
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-005"},  
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-006"},  
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-007"}  
  ]  
}
```

To load the data for the CUSTOMER table using the manifest file

1. Open the file customer-fw-manifest in a text editor.
2. Replace <your-bucket-name> with the name of your bucket.
3. Save the file.
4. Upload the file to the load folder on your bucket.
5. Run the following COPY command.

```
copy customer from 's3://<your-bucket-name>/load/customer-fw-manifest'  
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'  
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,  
c_region :12, c_phone:15,c_mktsegment:10'  
maxerror 10  
acceptinvchars as '^'  
manifest;
```

Load the DWDATE table using DATEFORMAT

In this step, you use the DELIMITER and DATEFORMAT options to load the DWDATE table.

When loading DATE and TIMESTAMP columns, COPY expects the default format, which is YYYY-MM-DD for dates and YYYY-MM-DD HH:MI:SS for timestamps. If the load data does not use a default format, you can use DATEFORMAT and TIMEFORMAT to specify the format.

The following excerpt shows date formats in the DWDATE table. Notice that the date formats in column two are inconsistent.

```
19920104 1992-01-04      Sunday January 1992 199201 Jan1992 1 4 4 1...
19920112 January 12, 1992 Monday January 1992 199201 Jan1992 2 12 12 1...
19920120 January 20, 1992 Tuesday January 1992 199201 Jan1992 3 20 20 1...
```

DATEFORMAT

You can specify only one date format. If the load data contains inconsistent formats, possibly in different columns, or if the format is not known at load time, you use DATEFORMAT with the 'auto' argument. When 'auto' is specified, COPY recognizes any valid date or time format and convert it to the default format. The 'auto' option recognizes several formats that are not supported when using a DATEFORMAT and TIMEFORMAT string. For more information, see [Using automatic recognition with DATEFORMAT and TIMEFORMAT \(p. 725\)](#).

To load the DWDATE table, run the following COPY command.

```
copy dwdate from 's3://<your-bucket-name>/load/dwdate-tab.tbl'
credentials 'aws_iam_role=arn:aws:iam:<aws-account-id>:role/<role-name>'
delimiter '\t'
dateformat 'auto';
```

Load the LINEORDER table using multiple files

This step uses the GZIP and COMPUPDATE options to load the LINEORDER table.

In this exercise, you load the LINEORDER table from a single data file and then load it again from multiple files. Doing this enables you to compare the load times for the two methods.

Note

The files for loading the LINEORDER table are provided in an AWS sample bucket. You don't need to upload files for this step.

GZIP, LZOP and BZIP2

You can compress your files using either gzip, lzop, or bzip2 compression formats. When loading from compressed files, COPY uncompresses the files during the load process. Compressing your files saves storage space and shortens upload times.

COMPUPDATE

When COPY loads an empty table with no compression encodings, it analyzes the load data to determine the optimal encodings. It then alters the table to use those encodings before beginning the load. This analysis process takes time, but it occurs, at most, once per table. To save time, you can skip this step by turning COMPUPDATE off. To enable an accurate evaluation of COPY times, you turn COMPUPDATE off for this step.

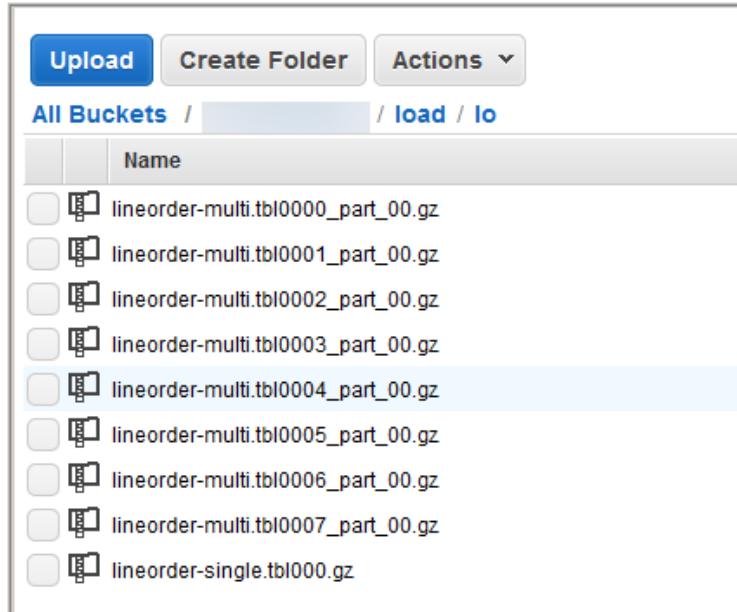
Multiple Files

The COPY command can load data very efficiently when it loads from multiple files in parallel instead of from a single file. You can split your data into files so that the number of files is a multiple of the number of slices in your cluster. If you do, Amazon Redshift divides the workload and distributes the data evenly among the slices. The number of slices per node depends on the node size of the cluster. For more information about the number of slices that each node size has, go to [About clusters and nodes](#) in the *Amazon Redshift Management Guide*.

For example, the dc2.large compute nodes used in this tutorial have two slices each, so the four-node cluster has eight slices. In previous steps, the load data was contained in eight files, even though the files are very small. In this step, you compare the time difference between loading from a single large file and loading from multiple files.

The files you use for this tutorial contain about 15 million records and occupy about 1.2 GB. These files are very small in Amazon Redshift scale, but sufficient to demonstrate the performance advantage of loading from multiple files. The files are large enough that the time required to download them and then upload them to Amazon S3 is excessive for this tutorial. Thus, you load the files directly from an AWS sample bucket.

The following screenshot shows the data files for LINEORDER.



To evaluate the performance of COPY with multiple files

1. Run the following command to COPY from a single file. Do not change the bucket name.

```
copy lineorder from 's3://awssampledbsampledb/load/lo/lineorder-single.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
gzip
compupdate off
region 'us-east-1';
```

2. Your results should be similar to the following. Note the execution time.

```
Warnings:
Load into table 'lineorder' completed, 14996734 record(s) loaded successfully.

0 row(s) affected.
copy executed successfully

Execution time: 51.56s
```

3. Run the following command to COPY from multiple files. Do not change the bucket name.

```
copy lineorder from 's3://awssampledbsampledb/load/lo/lineorder-multi.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
gzip
```

```
compupdate off
region 'us-east-1';
```

4. Your results should be similar to the following. Note the execution time.

```
Warnings:
Load into table 'lineorder' completed, 14996734 record(s) loaded successfully.

0 row(s) affected.
copy executed successfully

Execution time: 17.7s
```

5. Compare execution times.

In our example, the time to load 15 million records decreased from 51.56 seconds to 17.7 seconds, a reduction of 65.7 percent.

These results are based on using a four-node cluster. If your cluster has more nodes, the time savings is multiplied. For typical Amazon Redshift clusters, with tens to hundreds of nodes, the difference is even more dramatic. If you have a single node cluster, there is little difference between the execution times.

Next step

[Step 6: Vacuum and analyze the database \(p. 142\)](#)

Step 6: Vacuum and analyze the database

Whenever you add, delete, or modify a significant number of rows, you should run a VACUUM command and then an ANALYZE command. A *vacuum* recovers the space from deleted rows and restores the sort order. The ANALYZE command updates the statistics metadata, which enables the query optimizer to generate more accurate query plans. For more information, see [Vacuuming tables \(p. 110\)](#).

If you load the data in sort key order, a vacuum is fast. In this tutorial, you added a significant number of rows, but you added them to empty tables. That being the case, there is no need to resort, and you didn't delete any rows. COPY automatically updates statistics after loading an empty table, so your statistics should be up-to-date. However, as a matter of good housekeeping, you complete this tutorial by vacuuming and analyzing your database.

To vacuum and analyze the database, run the following commands.

```
vacuum;
analyze;
```

Next step

[Step 7: Clean up your resources \(p. 142\)](#)

Step 7: Clean up your resources

Your cluster continues to accrue charges as long as it is running. When you have completed this tutorial, you should return your environment to the previous state by following the steps in [Step 5: Revoke access and delete your sample cluster](#) in the *Amazon Redshift Getting Started Guide*.

If you want to keep the cluster, but recover the storage used by the SSB tables, run the following commands.

```
drop table part;
drop table supplier;
drop table customer;
drop table dwdate;
drop table lineorder;
```

Next

[Summary \(p. 143\)](#)

Summary

In this tutorial, you uploaded data files to Amazon S3 and then used COPY commands to load the data from the files into Amazon Redshift tables.

You loaded data using the following formats:

- Character-delimited
- CSV
- Fixed-width

You used the STL_LOAD_ERRORS system table to troubleshoot load errors, and then used the REGION, MANIFEST, MAXERROR, ACCEPTINVCHARS, DATEFORMAT, and NULL AS options to resolve the errors.

You applied the following best practices for loading data:

- [Use a COPY command to load data \(p. 18\)](#)
- [Loading data files \(p. 18\)](#)
- [Use a single COPY command to load from multiple files \(p. 18\)](#)
- [Compressing your data files \(p. 19\)](#)
- [Verify data files before and after a load \(p. 19\)](#)

For more information about Amazon Redshift best practices, see the following links:

- [Amazon Redshift best practices for loading data \(p. 17\)](#)
- [Amazon Redshift best practices for designing tables \(p. 15\)](#)
- [Amazon Redshift best practices for designing queries \(p. 21\)](#)

Unloading data

Topics

- [Unloading data to Amazon S3 \(p. 144\)](#)
- [Unloading encrypted data files \(p. 146\)](#)
- [Unloading data in delimited or fixed-width format \(p. 148\)](#)
- [Reloading unloaded data \(p. 149\)](#)

To unload data from database tables to a set of files in an Amazon S3 bucket, you can use the [UNLOAD \(p. 991\)](#) command with a SELECT statement. You can unload text data in either delimited format or fixed-width format, regardless of the data format that was used to load it. You can also specify whether to create compressed GZIP files.

You can limit the access users have to your Amazon S3 bucket by using temporary security credentials.

Unloading data to Amazon S3

Amazon Redshift splits the results of a select statement across a set of files, one or more files per node slice, to simplify parallel reloading of the data. Alternatively, you can specify that [UNLOAD \(p. 991\)](#) should write the results serially to one or more files by adding the PARALLEL OFF option. You can limit the size of the files in Amazon S3 by specifying the MAXFILESIZE parameter. UNLOAD automatically encrypts data files using Amazon S3 server-side encryption (SSE-S3).

You can use any select statement in the UNLOAD command that Amazon Redshift supports, except for a select that uses a LIMIT clause in the outer select. For example, you can use a select statement that includes specific columns or that uses a where clause to join multiple tables. If your query contains quotation marks (enclosing literal values, for example), you need to escape them in the query text (`\`). For more information, see the [SELECT \(p. 928\)](#) command reference. For more information about using a LIMIT clause, see the [Usage notes \(p. 998\)](#) for the UNLOAD command.

For example, the following UNLOAD command sends the contents of the VENUE table to the Amazon S3 bucket s3://mybucket/ticket/unload/.

```
unload ('select * from venue')
to 's3://mybucket/ticket/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

The file names created by the previous example include the prefix 'venue_'.

```
venue_0000_part_00
venue_0001_part_00
venue_0002_part_00
venue_0003_part_00
```

By default, UNLOAD writes data in parallel to multiple files, according to the number of slices in the cluster. To write data to a single file, specify PARALLEL OFF. UNLOAD writes the data serially, sorted absolutely according to the ORDER BY clause, if one is used. The maximum size for a data file is 6.2 GB. If the data size is greater than the maximum, UNLOAD creates additional files, up to 6.2 GB each.

The following example writes the contents VENUE to a single file. Only one file is required because the file size is less than 6.2 GB.

```
unload ('select * from venue')
to 's3://mybucket/ticket/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off;
```

Note

The UNLOAD command is designed to use parallel processing. We recommend leaving PARALLEL enabled for most cases, especially if the files will be used to load tables using a COPY command.

Assuming the total data size for VENUE is 5 GB, the following example writes the contents of VENUE to 50 files, each 100 MB in size.

```
unload ('select * from venue')
to 's3://mybucket/ticket/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off
maxfilesize 100 mb;
```

If you include a prefix in the Amazon S3 path string, UNLOAD will use that prefix for the file names.

```
unload ('select * from venue')
to 's3://mybucket/ticket/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

You can create a manifest file that lists the unload files by specifying the MANIFEST option in the UNLOAD command. The manifest is a text file in JSON format that explicitly lists the URL of each file that was written to Amazon S3.

The following example includes the manifest option.

```
unload ('select * from venue')
to 's3://mybucket/ticket/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

The following example shows a manifest for four unload files.

```
{
  "entries": [
    {"url":"s3://mybucket/ticket/venue_0000_part_00"},  

    {"url":"s3://mybucket/ticket/venue_0001_part_00"},  

    {"url":"s3://mybucket/ticket/venue_0002_part_00"},  

    {"url":"s3://mybucket/ticket/venue_0003_part_00"}
  ]
}
```

The manifest file can be used to load the same files by using a COPY with the MANIFEST option. For more information, see [Using a manifest to specify data files \(p. 71\)](#).

After you complete an UNLOAD operation, confirm that the data was unloaded correctly by navigating to the Amazon S3 bucket where UNLOAD wrote the files. You will see one or more numbered files per slice, starting with the number zero. If you specified the MANIFEST option, you will also see a file ending with 'manifest'. For example:

```
mybucket/ticket/venue_0000_part_00
mybucket/ticket/venue_0001_part_00
mybucket/ticket/venue_0002_part_00
mybucket/ticket/venue_0003_part_00
mybucket/ticket/venue_manifest
```

You can programmatically get a list of the files that were written to Amazon S3 by calling an Amazon S3 list operation after the UNLOAD completes. You can also query `STL_UNLOAD_LOG`.

The following query returns the pathname for files that were created by an UNLOAD. The [PG_LAST_QUERY_ID \(p. 1401\)](#) function returns the most recent query.

```
select query, substring(path,0,40) as path
from stl_unload_log
where query=2320
order by path;

query |          path
-----+-----
2320 | s3://my-bucket/venue0000_part_00
2320 | s3://my-bucket/venue0001_part_00
2320 | s3://my-bucket/venue0002_part_00
2320 | s3://my-bucket/venue0003_part_00
(4 rows)
```

If the amount of data is very large, Amazon Redshift might split the files into multiple parts per slice. For example:

```
venue_0000_part_00
venue_0000_part_01
venue_0000_part_02
venue_0001_part_00
venue_0001_part_01
venue_0001_part_02
...
```

The following UNLOAD command includes a quoted string in the select statement, so the quotation marks are escaped (`=\'0H\'`).

```
unload ('select venuename, venuecity from venue where venuestate=\'OH\' ')
to 's3://mybucket/ticket/venue/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

By default, UNLOAD will fail rather than overwrite existing files in the destination bucket. To overwrite the existing files, including the manifest file, specify the ALLOWOVERWRITE option.

```
unload ('select * from venue')
to 's3://mybucket/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest
allowoverwrite;
```

Unloading encrypted data files

UNLOAD automatically creates files using Amazon S3 server-side encryption with AWS-managed encryption keys (SSE-S3). You can also specify server-side encryption with an AWS Key Management

Service key (SSE-KMS) or client-side encryption with a customer managed key. UNLOAD doesn't support Amazon S3 server-side encryption using a customer managed key. For more information, see [Protecting data using server-side encryption](#).

To unload to Amazon S3 using server-side encryption with an AWS KMS key, use the KMS_KEY_ID parameter to provide the key ID as shown in the following example.

```
unload ('select venuename, venuecity from venue')
to 's3://mybucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
KMS_KEY_ID '1234abcd-12ab-34cd-56ef-1234567890ab'
encrypted;
```

If you want to provide your own encryption key, you can create client-side encrypted data files in Amazon S3 by using the UNLOAD command with the ENCRYPTED option. UNLOAD uses the same envelope encryption process that Amazon S3 client-side encryption uses. You can then use the COPY command with the ENCRYPTED option to load the encrypted files.

The process works like this:

1. You create a base64 encoded 256-bit AES key that you will use as your private encryption key, or *root symmetric key*.
2. You issue an UNLOAD command that includes your root symmetric key and the ENCRYPTED option.
3. UNLOAD generates a one-time-use symmetric key (called the *envelope symmetric key*) and an initialization vector (IV), which it uses to encrypt your data.
4. UNLOAD encrypts the envelope symmetric key using your root symmetric key.
5. UNLOAD then stores the encrypted data files in Amazon S3 and stores the encrypted envelope key and IV as object metadata with each file. The encrypted envelope key is stored as object metadata x-amz-meta-x-amz-key and the IV is stored as object metadata x-amz-meta-x-amz-iv.

For more information about the envelope encryption process, see the [Client-side data encryption with the AWS SDK for Java and Amazon S3](#) article.

To unload encrypted data files, add the root key value to the credentials string and include the ENCRYPTED option. If you use the MANIFEST option, the manifest file is also encrypted.

```
unload ('select venuename, venuecity from venue')
to 's3://mybucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
manifest
encrypted;
```

To unload encrypted data files that are GZIP compressed, include the GZIP option along with the root key value and the ENCRYPTED option.

```
unload ('select venuename, venuecity from venue')
to 's3://mybucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
encrypted gzip;
```

To load the encrypted data files, add the MASTER_SYMMETRIC_KEY parameter with the same root key value and include the ENCRYPTED option.

```
copy venue from 's3://mybucket/encrypted/venue_'
```

```
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'  
master_symmetric_key '<root_key>'  
encrypted;
```

Unloading data in delimited or fixed-width format

You can unload data in delimited format or fixed-width format. The default output is pipe-delimited (using the '|' character).

The following example specifies a comma as the delimiter:

```
unload ('select * from venue')  
to 's3://mybucket/ticket/venue/comma'  
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'  
delimiter ',';
```

The resulting output files look like this:

```
20,Air Canada Centre,Toronto,ON,0  
60,Rexall Place,Edmonton,AB,0  
100,U.S. Cellular Field,Chicago,IL,40615  
200,Al Hirschfeld Theatre,New York City,NY,0  
240,San Jose Repertory Theatre,San Jose,CA,0  
300,Kennedy Center Opera House,Washington,DC,0  
...
```

To unload the same result set to a tab-delimited file, issue the following command:

```
unload ('select * from venue')  
to 's3://mybucket/ticket/venue/tab'  
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'  
delimiter as '\t';
```

Alternatively, you can use a `FIXEDWIDTH` specification. This specification consists of an identifier for each table column and the width of the column (number of characters). The `UNLOAD` command will fail rather than truncate data, so specify a width that is at least as long as the longest entry for that column. Unloading fixed-width data works similarly to unloading delimited data, except that the resulting output contains no delimiting characters. For example:

```
unload ('select * from venue')  
to 's3://mybucket/ticket/venue/fw'  
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'  
fixedwidth '0:3,1:100,2:30,3:2,4:6';
```

The fixed-width output looks like this:

20	Air Canada Centre	Toronto	ON0
60	Rexall Place	Edmonton	AB0
100	U.S. Cellular Field	Chicago	IL40615
200	Al Hirschfeld Theatre	New York City	NY0
240	San Jose Repertory Theatre	San Jose	CA0
300	Kennedy Center Opera House	Washington	DC0

For more details about `FIXEDWIDTH` specifications, see the [UNLOAD \(p. 991\)](#) command.

Reloading unloaded data

To reload the results of an unload operation, you can use a COPY command.

The following example shows a simple case in which the VENUE table is unloaded using a manifest file, truncated, and reloaded.

```
unload ('select * from venue order by venueid')
to 's3://mybucket/ticket/venue/reload_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest
delimiter '|';

truncate venue;

copy venue
from 's3://mybucket/ticket/venue/reload_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest
delimiter '|';
```

After it is reloaded, the VENUE table looks like this:

```
select * from venue order by venueid limit 5;

venueid | venuename | venuecity | venuestate | venueseats
-----+-----+-----+-----+-----+
 1 | Toyota Park | Bridgeview | IL | 0
 2 | Columbus Crew Stadium | Columbus | OH | 0
 3 | RFK Stadium | Washington | DC | 0
 4 | CommunityAmerica Ballpark | Kansas City | KS | 0
 5 | Gillette Stadium | Foxborough | MA | 68756
(5 rows)
```

Creating user-defined functions

You can create a custom scalar user-defined function (UDF) using either a SQL SELECT clause or a Python program. The new function is stored in the database and is available for any user with sufficient privileges to run. You run a custom scalar UDF in much the same way as you run existing Amazon Redshift functions.

For Python UDFs, in addition to using the standard Python functionality, you can import your own custom Python modules. For more information, see [Python language support for UDFs \(p. 153\)](#).

You can also create AWS Lambda UDFs that use custom functions defined in Lambda as part of your SQL queries. Lambda UDFs enable you to write complex UDFs and integrate with third-party components. They also can help you overcome some of the limitations of current Python and SQL UDFs. For example, they can help you access network and storage resources and write more full-fledged SQL statements. You can create Lambda UDFs in any of the programming languages supported by Lambda, such as Java, Go, PowerShell, Node.js, C#, Python, and Ruby. Or you can use a custom runtime.

By default, all users can run UDFs. For more information about privileges, see [UDF security and privileges \(p. 150\)](#).

Topics

- [UDF security and privileges \(p. 150\)](#)
- [Creating a scalar SQL UDF \(p. 151\)](#)
- [Creating a scalar Python UDF \(p. 152\)](#)
- [Creating a scalar Lambda UDF \(p. 157\)](#)
- [Example uses of user-defined functions \(UDFs\) \(p. 163\)](#)

UDF security and privileges

To create a UDF, you must have permission for usage on language for SQL or plpythonu (Python). By default, USAGE ON LANGUAGE SQL is granted to PUBLIC, but you must explicitly grant USAGE ON LANGUAGE PLPYTHONU to specific users or groups.

To revoke usage for SQL, first revoke usage from PUBLIC. Then grant usage on SQL only to the specific users or groups permitted to create SQL UDFs. The following example revokes usage on SQL from PUBLIC. Then it grants usage to the user group udf_devs.

```
revoke usage on language sql from PUBLIC;
grant usage on language sql to group udf_devs;
```

To run a UDF, you must have permission to do so for each function. By default, permission to run new UDFs is granted to PUBLIC. To restrict usage, revoke this permission from PUBLIC for the function. Then grant the privilege to specific individuals or groups.

The following example revokes execution on function f_py_greater from PUBLIC. Then it grants usage to the user group udf_devs.

```
revoke execute on function f_py_greater(a float, b float) from PUBLIC;
grant execute on function f_py_greater(a float, b float) to group udf_devs;
```

Superusers have all privileges by default.

For more information, see [GRANT \(p. 887\)](#) and [REVOKE \(p. 916\)](#).

Creating a scalar SQL UDF

A scalar SQL UDF incorporates a SQL SELECT clause that runs when the function is called and returns a single value. The [CREATE FUNCTION \(p. 788\)](#) command defines the following parameters:

- (Optional) Input arguments. Each argument must have a data type.
- One return data type.
- One SQL SELECT clause. In the SELECT clause, refer to the input arguments using \$1, \$2, and so on, according to the order of the arguments in the function definition.

The input and return data types can be any standard Amazon Redshift data type.

Don't include a FROM clause in your SELECT clause. Instead, include the FROM clause in the SQL statement that calls the SQL UDF.

The SELECT clause can't include any of the following types of clauses:

- FROM
- INTO
- WHERE
- GROUP BY
- ORDER BY
- LIMIT

Scalar SQL function example

The following example creates a function that compares two numbers and returns the larger value. For more information, see [CREATE FUNCTION \(p. 788\)](#).

```
create function f_sql_greater (float, float)
    returns float
stable
as $$
    select case when $1 > $2 then $1
                else $2
            end
$$ language sql;
```

The following query calls the new f_sql_greater function to query the SALES table and return either COMMISSION or 20 percent of PRICEPAID, whichever is greater.

```
select f_sql_greater(commission, pricepaid*0.20) from sales;
```

Creating a scalar Python UDF

A scalar Python UDF incorporates a Python program that runs when the function is called and returns a single value. The [CREATE FUNCTION \(p. 788\)](#) command defines the following parameters:

- (Optional) Input arguments. Each argument must have a name and a data type.
- One return data type.
- One executable Python program.

The input and return data types can be SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, or TIMESTAMP. In addition, Python UDFs can use the data type ANYELEMENT, which Amazon Redshift automatically converts to a standard data type based on the arguments supplied at runtime. For more information, see [ANYELEMENT data type \(p. 153\)](#)

When an Amazon Redshift query calls a scalar UDF, the following steps occur at runtime:

1. The function converts the input arguments to Python data types.

For a mapping of Amazon Redshift data types to Python data types, see [Python UDF data types \(p. 152\)](#).

2. The function runs the Python program, passing the converted input arguments.
3. The Python code returns a single value. The data type of the return value must correspond to the RETURNS data type specified by the function definition.
4. The function converts the Python return value to the specified Amazon Redshift data type, then returns that value to the query.

Scalar Python UDF example

The following example creates a function that compares two numbers and returns the larger value. Note that the indentation of the code between the double dollar signs (\$\$) is a Python requirement. For more information, see [CREATE FUNCTION \(p. 788\)](#).

```
create function f_py_greater (a float, b float)
    returns float
stable
as $$ 
    if a > b:
        return a
    return b
$$ language plpython;
```

The following query calls the new f_greater function to query the SALES table and return either COMMISSION or 20 percent of PRICEPAID, whichever is greater.

```
select f_py_greater (commission, pricepaid*0.20) from sales;
```

Python UDF data types

Python UDFs can use any standard Amazon Redshift data type for the input arguments and the function's return value. In addition to the standard data types, UDFs support the data type *ANYELEMENT*, which Amazon Redshift automatically converts to a standard data type based on the arguments supplied at runtime. Scalar UDFs can return a data type of *ANYELEMENT*. For more information, see [ANYELEMENT data type \(p. 153\)](#).

During execution, Amazon Redshift converts the arguments from Amazon Redshift data types to Python data types for processing. It then converts the return value from the Python data type to the corresponding Amazon Redshift data type. For more information about Amazon Redshift data types, see [Data types \(p. 574\)](#).

The following table maps Amazon Redshift data types to Python data types.

Amazon Redshift data type	Python data type
smallint	int
integer	
bigint	
short	
long	
decimal or numeric	decimal
double	float
real	
boolean	bool
char	string
varchar	
timestamp	datetime

ANYELEMENT data type

ANYELEMENT is a *polymorphic data type*. This means that if a function is declared using ANYELEMENT for an argument's data type, the function can accept any standard Amazon Redshift data type as input for that argument when the function is called. The ANYELEMENT argument is set to the data type actually passed to it when the function is called.

If a function uses multiple ANYELEMENT data types, they must all resolve to the same actual data type when the function is called. All ANYELEMENT argument data types are set to the actual data type of the first argument passed to an ANYELEMENT. For example, a function declared as `f_equal(anyelement, anyelement)` will take any two input values, so long as they are of the same data type.

If the return value of a function is declared as ANYELEMENT, at least one input argument must be ANYELEMENT. The actual data type for the return value is the same as the actual data type supplied for the ANYELEMENT input argument.

Python language support for UDFs

You can create a custom UDF based on the Python programming language. The [Python 2.7 standard library](#) is available for use in UDFs, with the exception of the following modules:

- ScrolledText
- Tix
- Tkinter

- tk
- turtle
- smtplib

In addition to the Python Standard Library, the following modules are part of the Amazon Redshift implementation:

- [numpy 1.8.2](#)
- [pandas 0.14.1](#)
- [python-dateutil 2.2](#)
- [pytz 2014.7](#)
- [scipy 0.12.1](#)
- [six 1.3.0](#)
- [wsgiref 0.1.2](#)

You can also import your own custom Python modules and make them available for use in UDFs by executing a [CREATE LIBRARY \(p. 793\)](#) command. For more information, see [Importing custom Python library modules \(p. 154\)](#).

Important

Amazon Redshift blocks all network access and write access to the file system through UDFs.

Importing custom Python library modules

You define scalar functions using Python language syntax. You can use the Python Standard Library modules and Amazon Redshift preinstalled modules. You can also create your own custom Python library modules and import the libraries into your clusters, or use existing libraries from Python or third parties.

You cannot create a library that contains a module with the same name as a Python Standard Library module or an Amazon Redshift preinstalled Python module. If an existing user-installed library uses the same Python package as a library you create, you must drop the existing library before installing the new library.

You must be a superuser or have USAGE ON LANGUAGE plpythonu privilege to install custom libraries; however, any user with sufficient privileges to create functions can use the installed libraries. You can query the [PG_LIBRARY \(p. 1675\)](#) system catalog to view information about the libraries installed on your cluster.

To import a custom Python module into your cluster

This section provides an example of importing a custom Python module into your cluster. To perform the steps in this section, you must have an Amazon S3 bucket, where you upload the library package. You then install the package in your cluster. For more information about creating buckets, go to [Creating a bucket](#) in the *Amazon Simple Storage Service User Guide*.

In this example, let's suppose that you create UDFs to work with positions and distances in your data. Connect to your Amazon Redshift cluster from a SQL client tool, and run the following commands to create the functions.

```
CREATE FUNCTION f_distance (x1 float, y1 float, x2 float, y2 float) RETURNS float IMMUTABLE
as $$ 
    def distance(x1, y1, x2, y2):
        import math
        return math.sqrt((y2 - y1) ** 2 + (x2 - x1) ** 2)

    return distance(x1, y1, x2, y2)
```

```
$$ LANGUAGE plpythonu;

CREATE FUNCTION f_within_range (x1 float, y1 float, x2 float, y2 float) RETURNS bool
IMMUTABLE AS $$
    def distance(x1, y1, x2, y2):
        import math
        return math.sqrt((y2 - y1) ** 2 + (x2 - x1) ** 2)

    return distance(x1, y1, x2, y2) < 20
$$ LANGUAGE plpythonu;
```

Note that a few lines of code are duplicated in the previous functions. This duplication is necessary because a UDF cannot reference the contents of another UDF, and both functions require the same functionality. However, instead of duplicating code in multiple functions, you can create a custom library and configure your functions to use it.

To do so, first create the library package by following these steps:

1. Create a folder named **geometry**. This folder is the top level package of the library.
2. In the **geometry** folder, create a file named `__init__.py`. Note that the file name contains two double underscore characters. This file indicates to Python that the package can be initialized.
3. Also in the **geometry** folder, create a folder named **trig**. This folder is the subpackage of the library.
4. In the **trig** folder, create another file named `__init__.py` and a file named `line.py`. In this folder, `__init__.py` indicates to Python that the subpackage can be initialized and that `line.py` is the file that contains library code.

Your folder and file structure should be the same as the following:

```
geometry/
    __init__.py
    trig/
        __init__.py
        line.py
```

For more information about package structure, go to [Modules](#) in the Python tutorial on the Python website.

5. The following code contains a class and member functions for the library. Copy and paste it into `line.py`.

```
class LineSegment:
    def __init__(self, x1, y1, x2, y2):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
    def angle(self):
        import math
        return math.atan2(self.y2 - self.y1, self.x2 - self.x1)
    def distance(self):
        import math
        return math.sqrt((self.y2 - self.y1) ** 2 + (self.x2 - self.x1) ** 2)
```

After you have created the package, do the following to prepare the package and upload it to Amazon S3.

1. Compress the contents of the **geometry** folder into a .zip file named **geometry.zip**. Do not include the **geometry** folder itself; only include the contents of the folder as shown following:

```
geometry.zip
  __init__.py
  trig/
    __init__.py
    line.py
```

2. Upload **geometry.zip** to your Amazon S3 bucket.

Important

If the Amazon S3 bucket does not reside in the same region as your Amazon Redshift cluster, you must use the REGION option to specify the region in which the data is located. For more information, see [CREATE LIBRARY \(p. 793\)](#).

3. From your SQL client tool, run the following command to install the library. Replace `<bucket_name>` with the name of your bucket, and replace `<access key id>` and `<secret key>` with an access key and secret access key from your AWS Identity and Access Management (IAM) user credentials.

```
CREATE LIBRARY geometry LANGUAGE plpythonu FROM 's3://<bucket_name>/geometry.zip'
CREDENTIALS 'aws_access_key_id=<access key id>;aws_secret_access_key=<secret key>';
```

After you install the library in your cluster, you need to configure your functions to use the library. To do this, run the following commands.

```
CREATE OR REPLACE FUNCTION f_distance (x1 float, y1 float, x2 float, y2 float) RETURNS
  float IMMUTABLE as $$
    from trig.line import LineSegment

    return LineSegment(x1, y1, x2, y2).distance()
$$ LANGUAGE plpythonu;

CREATE OR REPLACE FUNCTION f_within_range (x1 float, y1 float, x2 float, y2 float) RETURNS
  bool IMMUTABLE as $$
    from trig.line import LineSegment

    return LineSegment(x1, y1, x2, y2).distance() < 20
$$ LANGUAGE plpythonu;
```

In the preceding commands, `import trig/line` eliminates the duplicated code from the original functions in this section. You can reuse the functionality provided by this library in multiple UDFs. Note that to import the module, you only need to specify the path to the subpackage and module name (`trig/line`).

UDF constraints

Within the constraints listed in this topic, you can use UDFs anywhere you use the Amazon Redshift built-in scalar functions. For more information, see [SQL functions reference \(p. 1020\)](#).

Amazon Redshift Python UDFs have the following constraints:

- Python UDFs cannot access the network or read or write to the file system.
- The total size of user-installed Python libraries cannot exceed 100 MB.
- The number of Python UDFs that can run concurrently per cluster is limited to one-fourth of the total concurrency level for the cluster. For example, if the cluster is configured with a concurrency of 15, a maximum of three UDFs can run concurrently. After the limit is reached, UDFs are queued for execution within workload management queues. SQL UDFs don't have a concurrency limit. For more information, see [Implementing workload management \(p. 491\)](#).
- When using Python UDFs, Amazon Redshift doesn't support the SUPER and HLLSKETCH data types.

Creating a scalar Lambda UDF

Amazon Redshift can use custom functions defined in AWS Lambda as part of SQL queries. You can write scalar Lambda UDFs in any programming languages supported by Lambda, such as Java, Go, PowerShell, Node.js, C#, Python, and Ruby. Or you can use a custom runtime.

Lambda UDFs are defined and managed in Lambda, and you can control the access privileges to invoke these UDFs in Amazon Redshift. You can invoke multiple Lambda functions in the same query or invoke the same function multiple times.

Use Lambda UDFs in any clauses of the SQL statements where scalar functions are supported. You can also use Lambda UDFs in any SQL statement such as SELECT, UPDATE, INSERT, or DELETE.

Note

Using Lambda UDFs can incur additional charges from the Lambda service. Whether it does so depends on factors such as the numbers of Lambda requests (UDF invocations) and the total duration of the Lambda program execution. However, there is no additional charge to use Lambda UDFs in Amazon Redshift. For information about AWS Lambda pricing, see [AWS Lambda Pricing](#).

The number of Lambda requests varies depending on the specific SQL statement clause where the Lambda UDF is used. For example, suppose the function is used in a WHERE clause such as the following.

```
SELECT a, b FROM t1 WHERE lambda_multiply(a, b) = 64; SELECT a, b FROM t1 WHERE a*b = lambda_multiply(2, 32)
```

In this case, Amazon Redshift calls the first SELECT statement for each and calls the second SELECT statement only once.

However, using a UDF in the projection part of the query might only invoke the Lambda function once for every qualified or aggregated row in the result set. You can configure batching of multiple invocations of your Lambda function to improve performance and lower costs.

Registering a Lambda UDF

The [CREATE EXTERNAL FUNCTION \(p. 755\)](#) command creates the following parameters:

- (Optional) A list of arguments with data type.
- One return data type.
- One function name of the external function that is called by Amazon Redshift.
- One IAM role that the Amazon Redshift cluster is authorized to assume and call to Lambda.
- One Lambda function name that the Lambda UDF invokes.

For information about CREATE EXTERNAL FUNCTION, see [CREATE EXTERNAL FUNCTION \(p. 755\)](#).

The input and return data types for this function can be any standard Amazon Redshift data type.

Amazon Redshift ensures that the external function can send and receive batched arguments and results.

Managing Lambda UDF security and privileges

To create a Lambda UDF, make sure that you have permissions for usage on the LANGUAGE EXFUNC. You must explicitly grant USAGE ON LANGUAGE EXFUNC or revoke USAGE ON LANGUAGE EXFUNC to specific users, groups, or public.

The following example grants usage on EXFUNC to PUBLIC.

```
grant usage on language exfunc to PUBLIC;
```

The following example revokes usage on exfunc from PUBLIC and then grants usage to the user group lambda_udf_devs.

```
revoke usage on language exfunc from PUBLIC;
grant usage on language exfunc to group lambda_udf_devs;
```

To run a Lambda UDF, make sure that you have permission for each function called. By default, permission to run new Lambda UDFs is granted to PUBLIC. To restrict usage, revoke this permission from PUBLIC for the function. Then, grant the privilege to specific users or groups.

The following example revokes execution on the function exfunc_sum from PUBLIC. Then, it grants usage to the user group lambda_udf_devs.

```
revoke execute on function exfunc_sum(int, int) from PUBLIC;
grant execute on function exfunc_sum(int, int) to group lambda_udf_devs;
```

Superusers have all privileges by default.

For more information about granting and revoking privileges, see [GRANT \(p. 887\)](#) and [REVOKE \(p. 916\)](#).

Configuring the authorization parameter for Lambda UDFs

The CREATE EXTERNAL FUNCTION command requires authorization to invoke Lambda functions in AWS Lambda. To start authorization, specify an AWS Identity and Access Management (IAM) role when you run the CREATE EXTERNAL FUNCTION command. For more information about IAM roles, see [IAM roles](#) in the *IAM User Guide*.

If there is an existing IAM role with permissions to invoke Lambda functions attached to your cluster, then you can substitute your role Amazon Resource Name (ARN) in the IAM_ROLE parameter for the command. Following sections describe the steps for using an IAM role in the CREATE EXTERNAL FUNCTION command.

Creating an IAM role for Lambda

The IAM role requires permission to invoke Lambda functions. While creating the IAM role, provide the permission in one of the following ways:

- Attach the AWSLambdaRole policy on the **Attach permissions policy** page while creating an IAM role. The AWSLambdaRole policy grants permissions to invoke Lambda functions which is the minimal requirement. For more information and other policies, see [Identity-based IAM policies for AWS Lambda](#) in the *AWS Lambda Developer Guide*.
- Create your own custom policy to attach to your IAM role with the lambda:InvokeFunction permission of either all resources or a particular Lambda function with the ARN of that function. For more information on how to create a policy, see [Creating IAM policies](#) in the *IAM User Guide*.

The following example policy enables invoking Lambda on a particular Lambda function.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Invoke",
            "Effect": "Allow",
            "Action": [
```

```
        "lambda:InvokeFunction"
    ],
    "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-function"
}
]
```

For more information on resources for Lambda functions, see [Resources and conditions for Lambda actions](#) in the *IAM API Reference*.

After creating your custom policy with the required permissions, you can attach your policy to the IAM role on the **Attach permissions policy** page while creating an IAM role.

For steps to create an IAM role, see [Authorizing Amazon Redshift to access other AWS services on your behalf](#) in the *Amazon Redshift Management Guide*.

If you don't want to create a new IAM role, you can add the permissions mentioned previously to your existing IAM role.

Associating an IAM role with the cluster

Attach the IAM role to your cluster. You can add a role to a cluster or view the roles associated with a cluster by using the Amazon Redshift Management Console, CLI, or API. For more information, see [Associating an IAM Role With a Cluster](#) in the *Amazon Redshift Management Guide*.

Including the IAM role in the command

Include the IAM role ARN in the CREATE EXTERNAL FUNCTION command. When you create an IAM role, IAM returns an Amazon Resource Name (ARN) for the role. To specify an IAM role, provide the role ARN with the IAM_ROLE parameter. The following shows the syntax for the IAM_ROLE parameter.

```
IAM_ROLE 'arn:aws:iam::aws-account-id:role/role-name'
```

To invoke Lambda functions which reside in other accounts within the same Region, see [Chaining IAM roles in Amazon Redshift](#).

Using the JSON interface between Amazon Redshift and AWS Lambda

Amazon Redshift uses a common interface for all Lambda functions that Amazon Redshift communicates to.

The following table shows the list of input fields that the designated Lambda functions that you can expect for the JSON payload.

Field name	Description	Value range
request_id	A universally unique identifier (UUID) that uniquely identifies each invoke request.	A valid UUID.
cluster	The full Amazon Resource Name (ARN) of the cluster.	A valid cluster ARN.

Field name	Description	Value range
user	The name of the user that makes the call.	A valid user name.
database	The name of the database that the query is running on.	A valid database name.
external_function	The fully qualified name of the external function that makes the call.	A valid fully qualified function name.
query_id	The query ID of the query that is making the call.	A valid query ID.
num_records	The number of arguments in the payload.	A value of 1 - 2^64.
arguments	The data payload in the specified format.	The data in array format must be a JSON array. Each element is a record that is an array if the number of arguments is larger than 1. By using an array, Amazon Redshift preserves the order of the records in the payload.

The order of the JSON array determines the order of batch processing. The Lambda function must process the arguments iteratively and produce the exact number of records. The following is an example of a payload.

```
{
  "request_id" : "23FF1F97-F28A-44AA-AB67-266ED976BF40",
  "cluster" : 'arn:aws:redshift:xxxx',
  "user" : "adminuser",
  "database" : "db1",
  "external_function": "public.foo",
  "query_id" : 5678234,
  "num_records" : 4,
  "arguments" : [
    [ 1, 2 ],
    [ 3, null],
    null,
    [ 4, 6]
  ]
}
```

The return output of the Lambda function contains the following fields.

Field name	Description	Value range
success	The indication of success or failure for the function.	A value of "true" or "false".
error_msg	The error message if the success value is "false" (if the function fails); otherwise, this field is ignored.	A valid message.

Field name	Description	Value range
num_records	The number of records in the payload.	A value of 1 - 2^64.
results	The results of the call in the specified format.	N/A

The following is an example of the Lambda function output.

```
{
  "success": true,    // true indicates the call succeeded
  "error_msg" : "my function isn't working", // shall only exist when success != true
  "num_records": 4,      // number of records in this payload
  "results" : [
    1,
    4,
    null,
    7
  ]
}
```

When you call Lambda functions from SQL queries, Amazon Redshift ensures the security of the connection with the following considerations:

- GRANT and REVOKE permissions. For more information about UDF security and privileges, see [UDF security and privileges \(p. 150\)](#).
- Amazon Redshift only submits the minimum set of data to the designated Lambda function.
- Amazon Redshift only calls the designated Lambda function with the designated IAM role.

Naming UDFs

You can avoid potential conflicts and unexpected results considering your UDF naming conventions before implementation. Because function names can be overloaded, they can collide with existing and future Amazon Redshift function names. This topic discusses overloading and presents a strategy for avoiding conflict.

Overloading function names

A function is identified by its name and *signature*, which is the number of input arguments and the data types of the arguments. Two functions in the same schema can have the same name if they have different signatures. In other words, the function names can be *overloaded*.

When you run a query, the query engine determines which function to call based on the number of arguments you provide and the data types of the arguments. You can use overloading to simulate functions with a variable number of arguments, up to the limit allowed by the [CREATE FUNCTION \(p. 788\)](#) command.

Preventing UDF naming conflicts

We recommend that you name all UDFs using the prefix f_. Amazon Redshift reserves the f_ prefix exclusively for UDFs and by prefixing your UDF names with f_, you ensure that your UDF name won't conflict with any existing or future Amazon Redshift built-in SQL function names. For example, by naming a new UDF f_sum, you avoid conflict with the Amazon Redshift SUM function. Similarly, if you

name a new function `f_fibonacci`, you avoid conflict if Amazon Redshift adds a function named `FIBONACCI` in a future release.

You can create a UDF with the same name and signature as an existing Amazon Redshift built-in SQL function without the function name being overloaded if the UDF and the built-in function exist in different schemas. Because built-in functions exist in the system catalog schema, `pg_catalog`, you can create a UDF with the same name in another schema, such as `public` or a user-defined schema. In some cases, you might call a function that is not explicitly qualified with a schema name. If so, Amazon Redshift searches the `pg_catalog` schema first by default. Thus, a built-in function runs before a new UDF with the same name.

You can change this behavior by setting the search path to place `pg_catalog` at the end. If you do so, your UDFs take precedence over built-in functions, but the practice can cause unexpected results. Adopting a unique naming strategy, such as using the reserved prefix `f_` is a more reliable practice. For more information, see [SET \(p. 977\)](#) and [search_path \(p. 1696\)](#).

Logging errors and warnings in UDFs

You can use the Python logging module to create user-defined error and warning messages in your UDFs. Following query execution, you can query the [SVL_UDF_LOG \(p. 1667\)](#) system view to retrieve logged messages.

Note

UDF logging consumes cluster resources and might affect system performance. We recommend implementing logging only for development and troubleshooting.

During query execution, the log handler writes messages to the `SVL_UDF_LOG` system view, along with the corresponding function name, node, and slice. The log handler writes one row to the `SVL_UDF_LOG` per message, per slice. Messages are truncated to 4096 bytes. The UDF log is limited to 500 rows per slice. When the log is full, the log handler discards older messages and adds a warning message to `SVL_UDF_LOG`.

Note

The Amazon Redshift UDF log handler escapes newlines (`\n`), pipe (`|`) characters, and backslash (`\`) characters with a backslash (`\`) character.

By default, the UDF log level is set to `WARNING`. Messages with a log level of `WARNING`, `ERROR`, and `CRITICAL` are logged. Messages with lower severity `INFO`, `DEBUG`, and `NOTSET` are ignored. To set the UDF log level, use the Python logger method. For example, the following sets the log level to `INFO`.

```
logger.setLevel(logging.INFO)
```

For more information about using the Python logging module, see [Logging facility for Python](#) in the Python documentation.

The following example creates a function named `f_pyerror` that imports the Python logging module, instantiates the logger, and logs an error.

```
CREATE OR REPLACE FUNCTION f_pyerror()
RETURNS INTEGER
VOLATILE AS
$$
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)
logger.info('Your info message here')
return 0
$$ language plpythonu;
```

The following example queries SVL_UDF_LOG to view the message logged in the previous example.

```
select funcname, node, slice, trim(message) as message
from svl_udf_log;

  funcname | query | node | slice |   message
-----+-----+-----+-----+
  f_pyerror | 12345 |     1|     1 | Your info message here
```

Example uses of user-defined functions (UDFs)

You can use user-defined functions to solve business problems by integrating Amazon Redshift with other components. Following are some examples of how others have used UDFs for their use cases:

- [Accessing external components using Amazon Redshift Lambda UDFs](#) – describes how Amazon Redshift Lambda UDFs work and walks through creating a Lambda UDF.
- [Translate and analyze text using SQL functions with Amazon Redshift, Amazon Translate, and Amazon Comprehend](#) – provides prebuilt Amazon Redshift Lambda UDFs that you can install with a few clicks to translate, redact, and analyze text fields.
- [Access Amazon Location Service from Amazon Redshift](#) – describes how to use Amazon Redshift Lambda UDFs to integrate with Amazon Location Service.
- [Data Tokenization with Amazon Redshift and Protegity](#) – describes how to integrate Amazon Redshift Lambda UDFs with the Protegity Serverless product.
- [Amazon Redshift UDFs](#) – a collection of Amazon Redshift SQL, Lambda, and Python UDFs.

Creating stored procedures in Amazon Redshift

You can define an Amazon Redshift stored procedure using the PostgreSQL procedural language PL/pgSQL to perform a set of SQL queries and logical operations. The procedure is stored in the database and available for any user with sufficient database privileges.

Unlike a user-defined function (UDF), a stored procedure can incorporate data definition language (DDL) and data manipulation language (DML) in addition to SELECT queries. A stored procedure doesn't need to return a value. You can use procedural language, including looping and conditional expressions, to control logical flow.

For details about SQL commands to create and manage stored procedures, see the following command topics:

- [CREATE PROCEDURE \(p. 815\)](#)
- [ALTER PROCEDURE \(p. 637\)](#)
- [DROP PROCEDURE \(p. 870\)](#)
- [SHOW PROCEDURE \(p. 987\)](#)
- [CALL \(p. 670\)](#)
- [GRANT \(p. 887\)](#)
- [REVOKE \(p. 916\)](#)
- [ALTER DEFAULT PRIVILEGES \(p. 631\)](#)

Topics

- [Overview of stored procedures in Amazon Redshift \(p. 164\)](#)
- [PL/pgSQL language reference \(p. 182\)](#)

Overview of stored procedures in Amazon Redshift

Stored procedures are commonly used to encapsulate logic for data transformation, data validation, and business-specific logic. By combining multiple SQL steps into a stored procedure, you can reduce round trips between your applications and the database.

For fine-grained access control, you can create stored procedures to perform functions without giving a user access to the underlying tables. For example, only the owner or a superuser can truncate a table, and a user needs write privileges to insert data into a table. Instead of granting a user privileges on the underlying tables, you can create a stored procedure that performs the task. You then give the user privileges to run the stored procedure.

A stored procedure with the DEFINER security attribute runs with the privileges of the stored procedure's owner. By default, a stored procedure has INVOKER security, which means the procedure uses the privileges of the user that calls the procedure.

To create a stored procedure, use the [CREATE PROCEDURE \(p. 815\)](#) command. To run a procedure, use the [CALL \(p. 670\)](#) command. Examples follow later in this section.

Note

Some clients might display the following error when creating an Amazon Redshift stored procedure.

```
ERROR: 42601: [Amazon](500310) unterminated dollar-quoted string at or near " $$
```

This error occurs due to the inability of the client to correctly parse the CREATE PROCEDURE statement with semicolons delimiting statements and with dollar sign (\$) quoting. This results in only a part of the statement sent to the Amazon Redshift server. You can often work around this error by using the Run as batch or Execute selected option of the client.

For example, when using an Aginity client, use the Run entire script as batch option. When you use SQL Workbench/J, we recommend version 124. When you use SQL Workbench/J version 125, consider specifying an alternate delimiter as a workaround.

CREATE PROCEDURE contains SQL statements delimited with a semicolon (;). Defining an alternate delimiter such as a slash (/) and placing it at the end of the CREATE PROCEDURE statement sends the statement to the Amazon Redshift server for processing. Following is an example.

```
CREATE OR REPLACE PROCEDURE test()
AS $$
BEGIN
    SELECT 1 a;
END;
$$
LANGUAGE plpgsql
;
/
```

For more information, see [Alternate delimiter](#) in the SQL Workbench/J documentation. Or use a client with better support for parsing CREATE PROCEDURE statements, such as the [query editor in the Amazon Redshift console](#) or TablePlus.

Topics

- [Naming stored procedures \(p. 166\)](#)
- [Security and privileges for stored procedures \(p. 167\)](#)
- [Returning a result set \(p. 168\)](#)
- [Managing transactions \(p. 169\)](#)
- [Trapping errors \(p. 178\)](#)
- [Logging stored procedures \(p. 181\)](#)
- [Considerations for stored procedure support \(p. 181\)](#)

The following example shows a procedure with no output arguments. By default, arguments are input (IN) arguments.

```
CREATE OR REPLACE PROCEDURE test_sp1(f1 int, f2 varchar)
AS $$
BEGIN
    RAISE INFO 'f1 = %, f2 = %', f1, f2;
END;
$$ LANGUAGE plpgsql;

call test_sp1(5, 'abc');
INFO: f1 = 5, f2 = abc
CALL
```

Note

When you write stored procedures, we recommend a best practice for securing sensitive values: Don't hardcode any sensitive information in stored procedure logic. For example, don't assign a user password in a CREATE USER statement in the body of a stored procedure. This poses a

security risk, because hardcoded values can be recorded as schema metadata in catalog tables. Instead, pass sensitive values, such as passwords, as arguments to the stored procedure, by means of parameters.

For more information about stored procedures, see [CREATE PROCEDURE \(p. 815\)](#) and [Creating stored procedures in Amazon Redshift \(p. 164\)](#). For more information about catalog tables, see [System catalog tables \(p. 1670\)](#).

The following example shows a procedure with output arguments. Arguments are input (IN), input and output (INOUT), and output (OUT).

```
CREATE OR REPLACE PROCEDURE test_sp2(f1 IN int, f2 INOUT varchar(256), out_var OUT
varchar(256))
AS $$

DECLARE
loop_var int;
BEGIN
IF f1 is null OR f2 is null THEN
    RAISE EXCEPTION 'input cannot be null';
END IF;
DROP TABLE if exists my_etl;
CREATE TEMP TABLE my_etl(a int, b varchar);
FOR loop_var IN 1..f1 LOOP
    insert into my_etl values (loop_var, f2);
    f2 := f2 || '+' || f2;
END LOOP;
SELECT INTO out_var count(*) from my_etl;
END;
$$ LANGUAGE plpgsql;

call test_sp2(2,'2019');

      f2      | column2
-----+-----
2019+2019+2019+2019 | 2
(1 row)
```

Naming stored procedures

If you define a procedure with the same name and different input argument data types or signature, you create a new procedure. As a result, the procedure name is overloaded. For more information, see [Overloading procedure names \(p. 166\)](#). Amazon Redshift doesn't enable procedure overloading based on output arguments. You can't have two procedures with the same name and input argument data types but different output argument types.

The owner or a superuser can replace the body of a stored procedure with a new one with the same signature. To change the signature or return types of a stored procedure, drop the stored procedure and recreate it. For more information, see [DROP PROCEDURE \(p. 870\)](#) and [CREATE PROCEDURE \(p. 815\)](#).

You can avoid potential conflicts and unexpected results by considering your naming conventions for stored procedures before implementing them. Because you can overload procedure names, they can collide with existing and future Amazon Redshift procedure names.

Overloading procedure names

A procedure is identified by its name and signature, which is the number of input arguments and the data types of the arguments. Two procedures in the same schema can have the same name if they have different signatures. In other words, you can overload procedure names.

When you run a procedure, the query engine determines which procedure to call based on the number of arguments that you provide and the data types of the arguments. You can use overloading to simulate

procedures with a variable number of arguments, up to the limit allowed by the CREATE PROCEDURE command. For more information, see [CREATE PROCEDURE \(p. 815\)](#).

Preventing naming conflicts

We recommend that you name all procedures using the prefix sp_. Amazon Redshift reserves the sp_ prefix exclusively for stored procedures. By prefixing your procedure names with sp_, you make sure that your procedure name won't conflict with any existing or future Amazon Redshift procedure name.

Security and privileges for stored procedures

By default, all users have privileges to create a procedure. To create a procedure, you must have USAGE privilege on the language PL/pgSQL, which is granted to PUBLIC by default. Only superusers and owners have the privilege to call a procedure by default. Superusers can run REVOKE USAGE on PL/pgSQL from a user if they want to prevent the user from creating a stored procedure.

To call a procedure, you must be granted EXECUTE privilege on the procedure. By default, EXECUTE privilege for new procedures is granted to the procedure owner and superusers. For more information, see [GRANT \(p. 887\)](#).

The user creating a procedure is the owner by default. The owner has CREATE, DROP, and EXECUTE privileges on the procedure by default. Superusers have all privileges.

The SECURITY attribute controls a procedure's privileges to access database objects. When you create a stored procedure, you can set the SECURITY attribute to either DEFINER or INVOKER. If you specify SECURITY INVOKER, the procedure uses the privileges of the user invoking the procedure. If you specify SECURITY DEFINER, the procedure uses the privileges of the owner of the procedure. INVOKER is the default.

Because a SECURITY DEFINER procedure runs with the privileges of the user that owns it, you must make sure that the procedure can't be misused. To make sure that SECURITY DEFINER procedures can't be misused, do the following:

- Grant EXECUTE on SECURITY DEFINER procedures to specific users, and not to PUBLIC.
- Qualify all database objects that the procedure must access with the schema names. For example, use myschema.mytable instead of just mytable.
- If you can't qualify an object name by its schema, set search_path when creating the procedure by using the SET option. Set search_path to exclude any schemas that are writable by untrusted users. This approach prevents any callers of this procedure from creating objects (for example, tables or views) that mask objects intended to be used by the procedure. For more information about the SET option, see [CREATE PROCEDURE \(p. 815\)](#).

The following example sets search_path to admin to ensure that the user_creds table is accessed from the admin schema and not from public or any other schema in the caller's search_path.

```
CREATE OR REPLACE PROCEDURE sp_get_credentials(userid int, o_creds OUT varchar)
AS $$
BEGIN
    SELECT creds INTO o_creds
    FROM user_creds
    WHERE user_id = $1;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER
-- Set a secure search_path
SET search_path = admin;
```

Returning a result set

You can return a result set using a cursor or a temp table.

Returning a cursor

To return a cursor, create a procedure with an INOUT argument defined with a `refcursor` data type. When you call the procedure, give the cursor a name. Then you can fetch the results from the cursor by name.

The following example creates a procedure named `get_result_set` with an INOUT argument named `rs_out` using the `refcursor` data type. The procedure opens the cursor using a `SELECT` statement.

```
CREATE OR REPLACE PROCEDURE get_result_set (param IN integer, rs_out INOUT refcursor)
AS $$$
BEGIN
  OPEN rs_out FOR SELECT * FROM fact_tbl where id >= param;
END;
$$ LANGUAGE plpgsql;
```

The following `CALL` command opens the cursor with the name `mycursor`. Use cursors only within transactions.

```
BEGIN;
CALL get_result_set(1, 'mycursor');
```

After the cursor is opened, you can fetch from the cursor, as the following example shows.

```
FETCH ALL FROM mycursor;

  id | secondary_id | name
-----+-----+-----
    1 |             1 | Joe
    1 |             2 | Ed
    2 |             1 | Mary
    1 |             3 | Mike
(4 rows)
```

In the end, the transaction is either committed or rolled back.

```
COMMIT;
```

A cursor returned by a stored procedure is subject to the same constraints and performance considerations as described in `DECLARE CURSOR`. For more information, see [Cursor constraints \(p. 856\)](#).

The following example shows the calling of the `get_result_set` stored procedure using a `refcursor` data type from JDBC. The literal '`mycursor`' (the name of the cursor) is passed to the `prepareStatement`. Then the results are fetched from the `ResultSet`.

```
static void refcursor_example(Connection conn) throws SQLException {
  conn.setAutoCommit(false);
  PreparedStatement proc = conn.prepareStatement("CALL get_result_set(1, 'mycursor')");
  proc.execute();
  ResultSet rs = statement.executeQuery("fetch all from mycursor");
  while (rs.next()) {
    int n = rs.getInt(1);
    System.out.println("n " + n);
```

}

Using a temp table

To return results, you can return a handle to a temp table containing result rows. The client can supply a name as a parameter to the stored procedure. Inside the stored procedure, dynamic SQL can be used to operate on the temp table. The following shows an example.

```
CREATE PROCEDURE get_result_set(param IN integer, tmp_name INOUT varchar(256)) as $$  
DECLARE  
    row record;  
BEGIN  
    EXECUTE 'drop table if exists ' || tmp_name;  
    EXECUTE 'create temp table ' || tmp_name || ' as select * from fact_tbl where id <= ' ||  
        param;  
END;  
$$ LANGUAGE plpgsql;  
  
CALL get_result_set(2, 'myresult');  
tmp_name  
-----  
myresult  
(1 row)  
  
SELECT * from myresult;  
id | secondary_id | name  
---+-----+  
1 | 1 | Joe  
2 | 1 | Mary  
1 | 2 | Ed  
1 | 3 | Mike  
(4 rows)
```

Managing transactions

You can create a stored procedure with default transaction management behavior or nonatomic behavior.

Default mode stored procedure transaction management

The default transaction mode automatic commit behavior causes each SQL command that runs separately to commit individually. A call to a stored procedure is treated as a single SQL command. The SQL statements inside a procedure behave as if they are in a transaction block that implicitly begins when the call starts and ends when the call finishes. A nested call to another procedure is treated like any other SQL statement and operates within the context of the same transaction as the caller. For more information about automatic commit behavior, see [Serializable isolation \(p. 119\)](#).

However, suppose that you call a stored procedure from within a user specified transaction block (defined by BEGIN...COMMIT). In this case, all statements in the stored procedure run in the context of the user-specified transaction. The procedure doesn't commit implicitly on exit. The caller controls the procedure commit or rollback.

If any error is encountered while running a stored procedure, all changes made in the current transaction are rolled back.

You can use the following transaction control statements in a stored procedure:

- COMMIT – commits all work done in the current transaction and implicitly begins a new transaction. For more information, see [COMMIT \(p. 676\)](#).

- ROLLBACK – rolls back the work done in the current transaction and implicitly begins a new transaction. For more information, see [ROLLBACK \(p. 927\)](#).

TRUNCATE is another statement that you can issue from within a stored procedure and influences transaction management. In Amazon Redshift, TRUNCATE issues a commit implicitly. This behavior stays the same in the context of stored procedures. When a TRUNCATE statement is issued from within a stored procedure, it commits the current transaction and begins a new one. For more information, see [TRUNCATE \(p. 990\)](#).

All statements that follow a COMMIT, ROLLBACK, or TRUNCATE statement run in the context of a new transaction. They do so until a COMMIT, ROLLBACK, or TRUNCATE statement is encountered or the stored procedure exits.

When you use a COMMIT, ROLLBACK, or TRUNCATE statement from within a stored procedure, the following constraints apply:

- If the stored procedure is called from within a transaction block, it can't issue a COMMIT, ROLLBACK, or TRUNCATE statement. This restriction applies within the stored procedure's own body and within any nested procedure call.
- If the stored procedure is created with SET config options, it can't issue a COMMIT, ROLLBACK, or TRUNCATE statement. This restriction applies within the stored procedure's own body and within any nested procedure call.
- Any cursor that is open (explicitly or implicitly) is closed automatically when a COMMIT, ROLLBACK, or TRUNCATE statement is processed. For constraints on explicit and implicit cursors, see [Considerations for stored procedure support \(p. 181\)](#).

Additionally, you can't run COMMIT or ROLLBACK using dynamic SQL. However, you can run TRUNCATE using dynamic SQL. For more information, see [Dynamic SQL \(p. 188\)](#).

When working with stored procedures, consider that the BEGIN and END statements in PL/pgSQL are only for grouping. They don't start or end a transaction. For more information, see [Block \(p. 183\)](#).

The following example demonstrates transaction behavior when calling a stored procedure from within an explicit transaction block. The two insert statements issued from outside the stored procedure and the one from within it are all part of the same transaction (3382). The transaction is committed when the user issues the explicit commit.

```
CREATE OR REPLACE PROCEDURE sp_insert_table_a(a int) LANGUAGE plpgsql
AS $$$
BEGIN
    INSERT INTO test_table_a values (a);
END;
$$;

Begin;
    insert into test_table_a values (1);
    Call sp_insert_table_a(2);
    insert into test_table_a values (3);
Commit;

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime , sequence;

    userid | xid | pid | type |           stmt_text
-----+-----+-----+-----+
    103 | 3382 | 599 | UTILITY | Begin;
    103 | 3382 | 599 | QUERY   | insert into test_table_a values (1);
    103 | 3382 | 599 | UTILITY | Call sp_insert_table_a(2);
    103 | 3382 | 599 | QUERY   | INSERT INTO test_table_a values ( $1 )
```

103 3382 599 QUERY insert into test_table_a values (3);
103 3382 599 UTILITY COMMIT

In contrast, take an example when the same statements are issued from outside of an explicit transaction block and the session has autocommit set to ON. In this case, each statement runs in its own transaction.

insert into test_table_a values (1);
Call sp_insert_table_a(2);
insert into test_table_a values (3);
select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime , sequence;
userid xid pid type
stmt_text
-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----
103 3388 599 QUERY insert into test_table_a values (1);
103 3388 599 UTILITY COMMIT
103 3389 599 UTILITY Call sp_insert_table_a(2);
103 3389 599 QUERY INSERT INTO test_table_a values (\$1)
103 3389 599 UTILITY COMMIT
103 3390 599 QUERY insert into test_table_a values (3);
103 3390 599 UTILITY COMMIT

The following example issues a TRUNCATE statement after inserting into test_table_a. The TRUNCATE statement issues an implicit commit that commits the current transaction (3335) and starts a new one (3336). The new transaction is committed when the procedure exits.

CREATE OR REPLACE PROCEDURE sp_truncate_proc(a int, b int) LANGUAGE plpgsql
AS \$\$
BEGIN
INSERT INTO test_table_a values (a);
TRUNCATE test_table_b;
INSERT INTO test_table_b values (b);
END;
\$\$;
Call sp_truncate_proc(1,2);
select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime , sequence;
userid xid pid type
stmt_text
-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----
103 3335 23636 UTILITY Call sp_truncate_proc(1,2);
103 3335 23636 QUERY INSERT INTO test_table_a values (\$1)
103 3335 23636 UTILITY TRUNCATE test_table_b
103 3335 23636 UTILITY COMMIT
103 3336 23636 QUERY INSERT INTO test_table_b values (\$1)
103 3336 23636 UTILITY COMMIT

The following example issues a TRUNCATE from a nested call. The TRUNCATE commits all work done so far in the outer and inner procedures in a transaction (3344). It starts a new transaction (3345). The new transaction is committed when the outer procedure exits.

CREATE OR REPLACE PROCEDURE sp_inner(c int, d int) LANGUAGE plpgsql
AS \$\$
BEGIN

```

    INSERT INTO inner_table values (c);
    TRUNCATE outer_table;
    INSERT INTO inner_table values (d);
END;
$$;

CREATE OR REPLACE PROCEDURE sp_outer(a int, b int, c int, d int) LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO outer_table values (a);
    Call sp_inner(c, d);
    INSERT INTO outer_table values (b);
END;
$$;

Call sp_outer(1, 2, 3, 4);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime , sequence;

userid | xid   | pid   | type   |           stmt_text
-----+-----+-----+-----+
103  | 3344  | 23636 | UTILITY | Call sp_outer(1, 2, 3, 4);
103  | 3344  | 23636 | QUERY   | INSERT INTO outer_table values ( $1 )
103  | 3344  | 23636 | UTILITY | CALL sp_inner( $1 , $2 )
103  | 3344  | 23636 | QUERY   | INSERT INTO inner_table values ( $1 )
103  | 3344  | 23636 | UTILITY | TRUNCATE outer_table
103  | 3344  | 23636 | UTILITY | COMMIT
103  | 3345  | 23636 | QUERY   | INSERT INTO inner_table values ( $1 )
103  | 3345  | 23636 | QUERY   | INSERT INTO outer_table values ( $1 )
103  | 3345  | 23636 | UTILITY | COMMIT

```

The following example shows that cursor `curl1` was closed when the `TRUNCATE` statement committed.

```

CREATE OR REPLACE PROCEDURE sp_open_cursor_truncate()
LANGUAGE plpgsql
AS $$
DECLARE
    rec RECORD;
    curl1 cursor for select * from test_table_a order by 1;
BEGIN
    open curl1;
    TRUNCATE table test_table_b;
    Loop
        fetch curl1 into rec;
        raise info '%', rec.c1;
        exit when not found;
    End Loop;
END
$$;

call sp_open_cursor_truncate();
ERROR: cursor "curl1" does not exist
CONTEXT: PL/pgSQL function "sp_open_cursor_truncate" line 8 at fetch

```

The following example issues a `TRUNCATE` statement and can't be called from within an explicit transaction block.

```

CREATE OR REPLACE PROCEDURE sp_truncate_atomic() LANGUAGE plpgsql
AS $$
BEGIN

```

```

    TRUNCATE test_table_b;
END;
$$;

Begin;
  Call sp_truncate_atomic();
ERROR: TRUNCATE cannot be invoked from a procedure that is executing in an atomic context.
HINT: Try calling the procedure as a top-level call i.e. not from within an explicit
      transaction block.
Or, if this procedure (or one of its ancestors in the call chain) was created with SET
      config options, recreate the procedure without them.
CONTEXT: SQL statement "TRUNCATE test_table_b"
PL/pgSQL function "sp_truncate_atomic" line 2 at SQL statement

```

The following example shows that a user who is not a superuser or the owner of a table can issue a TRUNCATE statement on the table. The user does this using a Security Definer stored procedure. The example shows the following actions:

- The user1 creates table test_tbl.
- The user1 creates stored procedure sp_truncate_test_tbl.
- The user1 grants EXECUTE privilege on the stored procedure to user2.
- The user2 runs the stored procedure to truncate table test_tbl. The example shows the row count before and after the TRUNCATE command.

```

set session_authorization to user1;
create table test_tbl(id int, name varchar(20));
insert into test_tbl values (1,'john'), (2, 'mary');
CREATE OR REPLACE PROCEDURE sp_truncate_test_tbl() LANGUAGE plpgsql
AS $$$
DECLARE
  tbl_rows int;
BEGIN
  select count(*) into tbl_rows from test_tbl;
  RAISE INFO 'RowCount before Truncate: %', tbl_rows;
  TRUNCATE test_tbl;
  select count(*) into tbl_rows from test_tbl;
  RAISE INFO 'RowCount after Truncate: %', tbl_rows;
END;
$$ SECURITY DEFINER;
grant execute on procedure sp_truncate_test_tbl() to user2;
reset session_authorization;

set session_authorization to user2;
call sp_truncate_test_tbl();
INFO: RowCount before Truncate: 2
INFO: RowCount after Truncate: 0
CALL
reset session_authorization;

```

The following example issues COMMIT twice. The first COMMIT commits all work done in transaction 10363 and implicitly starts transaction 10364. Transaction 10364 is committed by the second COMMIT statement.

```

CREATE OR REPLACE PROCEDURE sp_commit(a int, b int) LANGUAGE plpgsql
AS $$$
BEGIN
  INSERT INTO test_table values (a);
  COMMIT;
  INSERT INTO test_table values (b);

```

```

    COMMIT;
END;
$$;

call sp_commit(1,2);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime , sequence;
userid | xid | pid | type | stmt_text
-----+-----+-----+
100 | 10363 | 3089 | UTILITY | call sp_commit(1,2);
100 | 10363 | 3089 | QUERY | INSERT INTO test_table values ( $1 )
100 | 10363 | 3089 | UTILITY | COMMIT
100 | 10364 | 3089 | QUERY | INSERT INTO test_table values ( $1 )
100 | 10364 | 3089 | UTILITY | COMMIT

```

The following example issues a ROLLBACK statement if `sum_vals` is greater than 2. The first ROLLBACK statement rolls back all the work done in transaction 10377 and starts a new transaction 10378. Transaction 10378 is committed when the procedure exits.

```

CREATE OR REPLACE PROCEDURE sp_rollback(a int, b int) LANGUAGE plpgsql
AS $$$
DECLARE
    sum_vals int;
BEGIN
    INSERT INTO test_table values (a);
    SELECT sum(c1) into sum_vals from test_table;
    IF sum_vals > 2 THEN
        ROLLBACK;
    END IF;

    INSERT INTO test_table values (b);
END;
$$;

call sp_rollback(1, 2);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime , sequence;
userid | xid | pid | type | stmt_text
-----+-----+-----+
100 | 10377 | 3089 | UTILITY | call sp_rollback(1, 2);
100 | 10377 | 3089 | QUERY | INSERT INTO test_table values ( $1 )
100 | 10377 | 3089 | QUERY | SELECT sum(c1) from test_table
100 | 10377 | 3089 | QUERY | Undoing 1 transactions on table 133646 with current xid
10377 : 10377
100 | 10378 | 3089 | QUERY | INSERT INTO test_table values ( $1 )
100 | 10378 | 3089 | UTILITY | COMMIT

```

Nonatomic mode stored procedure transaction management

A stored procedure created in NONATOMIC mode has different transaction control behavior from a procedure created in default mode. Similar to the automatic commit behavior of SQL commands outside stored procedures, each SQL statement inside a NONATOMIC procedure runs in its own transaction and commits automatically. If a user begins an explicit transaction block within a NONATOMIC stored procedure, then the SQL statements within the block do not automatically commit. The transaction block controls commit or rollback of statements within it.

In NONATOMIC stored procedures, you can open an explicit transaction block inside the procedure using the START TRANSACTION statement. However, if there is already an open transaction block, this statement will do nothing because Amazon Redshift does not support sub transactions. The previous transaction continues.

When you work with cursor FOR loops inside a NONATOMIC procedure, make sure you open an explicit transaction block before iterating through the results of a query. Otherwise, the cursor is closed when the SQL statement inside the loop is automatically committed.

Some of the considerations when using NONATOMIC mode behavior are as follows:

- Each SQL statement inside the stored procedure is automatically committed if there is no open transaction block, and the session has autocommit set to ON.
- You can issue a COMMIT/ROLLBACK/TRUNCATE statement to end the transaction if the stored procedure is called from within a transaction block. This is not possible in default mode.
- You can issue a START TRANSACTION statement to begin a transaction block inside the stored procedure.

The following examples demonstrate transaction behavior when working with NONATOMIC stored procedures. The session for all the following examples has autocommit set to ON.

In the following example, a NONATOMIC stored procedure has two INSERT statements. When the procedure is called outside of a transaction block, every INSERT statement within the procedure automatically commits.

```

CREATE TABLE test_table_a(v int);
CREATE TABLE test_table_b(v int);

CREATE OR REPLACE PROCEDURE sp_nonatomic_insert_table_a(a int, b int) NONATOMIC AS
$$
BEGIN
    INSERT INTO test_table_a values (a);
    INSERT INTO test_table_b values (b);
END;
$$
LANGUAGE plpgsql;

Call sp_nonatomic_insert_table_a(1,2);

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime , sequence;

userid | xid | pid | type | stmt_text
-----+-----+-----+-----+
  1 | 1792 | 1073807554 | UTILITY | Call sp_nonatomic_insert_table_a(1,2);
  1 | 1792 | 1073807554 | QUERY | INSERT INTO test_table_a values ( $1 )
  1 | 1792 | 1073807554 | UTILITY | COMMIT
  1 | 1793 | 1073807554 | QUERY | INSERT INTO test_table_b values ( $1 )
  1 | 1793 | 1073807554 | UTILITY | COMMIT
(5 rows)

```

However, when the procedure is called from within a BEGIN..COMMIT block, all the statements are part of the same transaction (xid=1799).

```

Begin;
INSERT INTO test_table_a values (10);
Call sp_nonatomic_insert_table_a(20,30);
INSERT INTO test_table_b values (40);
Commit;

```

```
Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime , sequence;

+-----+-----+-----+-----+-----+
| userid | xid | pid | type | stmt_text |
+-----+-----+-----+-----+-----+
| 1 | 1799 | 1073914035 | UTILITY | Begin;
| 1 | 1799 | 1073914035 | QUERY | INSERT INTO test_table_a values (10);
| 1 | 1799 | 1073914035 | UTILITY | Call sp_nonatomic_insert_table_a(20,30);
| 1 | 1799 | 1073914035 | QUERY | INSERT INTO test_table_a values ( $1 )
| 1 | 1799 | 1073914035 | QUERY | INSERT INTO test_table_b values ( $1 )
| 1 | 1799 | 1073914035 | QUERY | INSERT INTO test_table_b values (40);
| 1 | 1799 | 1073914035 | UTILITY | COMMIT
+-----+-----+-----+-----+-----+
(7 rows)
```

In this example, two INSERT statements are between START TRANSACTION...COMMIT. When the procedure is called outside of a transaction block, the two INSERT statements are in the same transaction (xid=1866).

```
CREATE OR REPLACE PROCEDURE sp_nonatomic_txn_block(a int, b int) NONATOMIC AS
$$
BEGIN
    START TRANSACTION;
    INSERT INTO test_table_a values (a);
    INSERT INTO test_table_b values (b);
    COMMIT;
END;
$$
LANGUAGE plpgsql;

Call sp_nonatomic_txn_block(1,2);

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime , sequence;

+-----+-----+-----+-----+-----+
| userid | xid | pid | type | stmt_text |
+-----+-----+-----+-----+-----+
| 1 | 1865 | 1073823998 | UTILITY | Call sp_nonatomic_txn_block(1,2);
| 1 | 1866 | 1073823998 | QUERY | INSERT INTO test_table_a values ( $1 )
| 1 | 1866 | 1073823998 | QUERY | INSERT INTO test_table_b values ( $1 )
| 1 | 1866 | 1073823998 | UTILITY | COMMIT
+-----+-----+-----+-----+-----+
(4 rows)
```

When the procedure is called from within a BEGIN...COMMIT block, the START TRANSACTION inside the procedure does nothing because there is already an open transaction. The COMMIT inside the procedure commits the current transaction (xid=1876) and starts a new one.

```
Begin;
INSERT INTO test_table_a values (10);
Call sp_nonatomic_txn_block(20,30);
INSERT INTO test_table_b values (40);
Commit;

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime , sequence;

+-----+-----+-----+-----+-----+
| userid | xid | pid | type | stmt_text |
+-----+-----+-----+-----+-----+
| 1 | 1876 | 1073832133 | UTILITY | Begin;
| 1 | 1876 | 1073832133 | QUERY | INSERT INTO test_table_a values (10);
| 1 | 1876 | 1073832133 | UTILITY | Call sp_nonatomic_txn_block(20,30);
| 1 | 1876 | 1073832133 | QUERY | INSERT INTO test_table_a values ( $1 )
| 1 | 1876 | 1073832133 | QUERY | INSERT INTO test_table_b values ( $1 )
+-----+-----+-----+-----+-----+
```

```

1 | 1876 | 1073832133 | UTILITY | COMMIT
1 | 1878 | 1073832133 | QUERY   | INSERT INTO test_table_b values (40);
1 | 1878 | 1073832133 | UTILITY | COMMIT
(8 rows)

```

This example shows how to work with cursor loops. Table `test_table_a` has three values. The objective is to iterate through the three values and insert them into table `test_table_b`. If a NONATOMIC stored procedure is created in the following way, it will throw the error cursor "cur1" does not exist after executing `INSERT` statement in the first loop. This is because the auto commit of the `INSERT` closes the open cursor.

```

insert into test_table_a values (1), (2), (3);

CREATE OR REPLACE PROCEDURE sp_nonatomic_cursor() NONATOMIC
LANGUAGE plpgsql
AS $$$
DECLARE
    rec RECORD;
    cur1 cursor for select * from test_table_a order by 1;
BEGIN
    open cur1;
    Loop
        fetch cur1 into rec;
        exit when not found;
        raise info '%', rec.v;
        insert into test_table_b values (rec.v);
    End Loop;
END
$$;

CALL sp_nonatomic_cursor();

INFO: 1
ERROR: cursor "cur1" does not exist
CONTEXT: PL/pgSQL function "sp_nonatomic_cursor" line 7 at fetch

```

To make the cursor loop work, put it between `START TRANSACTION...COMMIT`.

```

insert into test_table_a values (1), (2), (3);

CREATE OR REPLACE PROCEDURE sp_nonatomic_cursor() NONATOMIC
LANGUAGE plpgsql
AS $$$
DECLARE
    rec RECORD;
    cur1 cursor for select * from test_table_a order by 1;
BEGIN
    START TRANSACTION;
    open cur1;
    Loop
        fetch cur1 into rec;
        exit when not found;
        raise info '%', rec.v;
        insert into test_table_b values (rec.v);
    End Loop;
    COMMIT;
END
$$;

CALL sp_nonatomic_cursor();

INFO: 1
INFO: 2

```

```
INFO: 3
CALL
```

Trapping errors

When a query or command in a stored procedure causes an error, subsequent queries don't run and the transaction is rolled back. But you can handle errors using an EXCEPTION block.

Note

The default behavior is that an error will cause subsequent queries not to run, even when there are no additional error-generating conditions in the stored procedure.

```
[ <>label> ]
[ DECLARE
  declarations ]
BEGIN
  statements
EXCEPTION
  WHEN OTHERS THEN
    statements
END;
```

When an exception occurs, and you add an exception-handling block, you can write RAISE statements and most other PL/pgSQL statements. For example, you can raise an exception with a custom message or insert a record into a logging table.

When entering the exception-handling block, the current transaction is rolled back and a new transaction is created to run the statements in the block. If the statements in the block run without error, the transaction is committed and the exception is re-thrown. Lastly, the stored procedure exits.

The only supported condition in an exception block is OTHERS, which matches every error type except query cancellation. Also, if an error occurs in an exception-handling block, it can be caught by an outer exception-handling block.

When an error occurs inside the NONATOMIC procedure, the error is not re-thrown if it is handled by an exception block. See the PL/pgSQL statement RAISE to throw an exception caught by the exception handling block. This statement is only valid in exception handling blocks. For more information see [RAISE \(p. 196\)](#).

There are several system tables available to help you gather information about various types of errors. For more information, see [STL_LOAD_ERRORS \(p. 1526\)](#), [STL_ERROR \(p. 1513\)](#), and [SYS_STREAM_SCAN_ERRORS \(p. 1490\)](#). There are also additional system tables you can use to troubleshoot errors. More information about these can be found at [System tables and views reference \(p. 1409\)](#).

Example

The following example shows how to write statements in the exception-handling block. The stored procedure is using default transaction management behavior.

```
CREATE TABLE employee (firstname varchar, lastname varchar);
INSERT INTO employee VALUES ('Tomas','Smith');
CREATE TABLE employee_error_log (message varchar);

CREATE OR REPLACE PROCEDURE update_employee_sp() AS
$$
BEGIN
  UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
```

```

EXECUTE 'select invalid';
EXCEPTION WHEN OTHERS THEN
    RAISE INFO 'An exception occurred.';
    INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
END;
$$
LANGUAGE plpgsql;

CALL update_employee_sp();

INFO: An exception occurred.
ERROR: column "invalid" does not exist
CONTEXT: SQL statement "select invalid"
PL/pgSQL function "update_employee_sp" line 3 at execute statement

```

In this example, if we call `update_employee_sp`, the informational message *An exception occurred.* is raised and the error message is inserted in the logging table's `employee_error_log` log. The original exception is thrown again before the stored procedure exits. The following queries show records that result from running the example.

```

SELECT * from employee;
firstname | lastname
-----+-----
Tomas     | Smith

SELECT * from employee_error_log;
message
-----
Error message: column "invalid" does not exist

```

For more information about `RAISE`, including formatting help and a list of additional levels, see [Supported PL/pgSQL statements \(p. 186\)](#).

The following example shows how to write statements in the exception-handling block. The stored procedure is using `NONATOMIC` transaction management behavior. In this example, there is no error thrown back to caller after the procedure call completes. The `UPDATE` statement is not rolled back due to the error in the next statement. The informational message is raised and the error message is inserted in the logging table.

```

CREATE TABLE employee (firstname varchar, lastname varchar);
INSERT INTO employee VALUES ('Tomas','Smith');
CREATE TABLE employee_error_log (message varchar);

-- Create the SP in NONATOMIC mode
CREATE OR REPLACE PROCEDURE update_employee_sp_2() NONATOMIC AS
$$
BEGIN
    UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
    EXECUTE 'select invalid';
EXCEPTION WHEN OTHERS THEN
    RAISE INFO 'An exception occurred.';
    INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
END;
$$
LANGUAGE plpgsql;

CALL update_employee_sp_2();
INFO: An exception occurred.
CALL

```

```

SELECT * from employee;
  firstname | lastname
  -----+-----
  Adam      | Smith
(1 row)

SELECT * from employee_error_log;
  message
  -----
  Error message: column "invalid" does not exist
(1 row)

```

This example shows how to create a procedure with two sub blocks. When the stored procedure is called, the error from the first sub block is handled by its exception handling block. After the first sub block completes, the procedure continues to execute the second sub block. You can see from the result that no error is thrown when the procedure call completes. The UPDATE and INSERT operations on table employee are committed. Error messages from both exception blocks are inserted in the logging table.

```

CREATE TABLE employee (firstname varchar, lastname varchar);
INSERT INTO employee VALUES ('Tomas','Smith');
CREATE TABLE employee_error_log (message varchar);

CREATE OR REPLACE PROCEDURE update_employee_sp_3() NONATOMIC AS
$$
BEGIN
    BEGIN
        UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
        EXECUTE 'select invalid1';
    EXCEPTION WHEN OTHERS THEN
        RAISE INFO 'An exception occurred in the first block.';
        INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
    END;
    BEGIN
        INSERT INTO employee VALUES ('Edie','Robertson');
        EXECUTE 'select invalid2';
    EXCEPTION WHEN OTHERS THEN
        RAISE INFO 'An exception occurred in the second block.';
        INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
    END;
END;
$$
LANGUAGE plpgsql;

CALL update_employee_sp_3();
INFO: An exception occurred in the first block.
INFO: An exception occurred in the second block.
CALL

SELECT * from employee;
  firstname | lastname
  -----+-----
  Adam      | Smith
  Edie     | Robertson
(2 rows)

SELECT * from employee_error_log;
  message
  -----
  Error message: column "invalid1" does not exist
  Error message: column "invalid2" does not exist

```

(2 rows)

Logging stored procedures

Details about stored procedures are logged in the following system tables and views:

- SVL_STORED_PROC_CALL – details are logged about the stored procedure call's start time and end time, and whether the call is ended before completion. For more information, see [SVL_STORED_PROC_CALL \(p. 1663\)](#).
- SVL_STORED_PROC_MESSAGES – messages in stored procedures emitted by the RAISE query are logged with the corresponding logging level. For more information, see [SVL_STORED_PROC_MESSAGES \(p. 1664\)](#).
- SVL_QLOG – the query ID of the procedure call is logged for each query called from a stored procedure. For more information, see [SVL_QLOG \(p. 1640\)](#).
- STL.UtilityText – stored procedure calls are logged after they are completed. For more information, see [STL.UtilityText \(p. 1567\)](#).
- PG_PROC_INFO – this system catalog view shows information about stored procedures. For more information, see [PG_PROC_INFO \(p. 1676\)](#).

Considerations for stored procedure support

The following considerations apply when you use Amazon Redshift stored procedures.

Differences between Amazon Redshift and PostgreSQL for stored procedure support

The following are differences between stored procedure support in Amazon Redshift and PostgreSQL:

- Amazon Redshift doesn't support subtransactions, and hence has limited support for exception handling blocks.

Considerations and limits

The following are considerations on stored procedures in Amazon Redshift:

- The maximum number of stored procedures for a database is 10,000.
- The maximum size of the source code for a procedure is 2 MB.
- The maximum number of explicit and implicit cursors that you can open concurrently in a user session is one. FOR loops that iterate over the result set of a SQL statement open implicit cursors. Nested cursors aren't supported.
- Explicit and implicit cursors have the same restrictions on the result set size as standard Amazon Redshift cursors. For more information, see [Cursor constraints \(p. 856\)](#).
- The maximum number of levels for nested calls is 16.
- The maximum number of procedure parameters is 32 for input arguments and 32 for output arguments.
- The maximum number of variables in a stored procedure is 1,024.
- Any SQL command that requires its own transaction context isn't supported inside a stored procedure. Examples include:
 - PREPARE
 - CREATE/DROP DATABASE

- CREATE EXTERNAL TABLE
- VACUUM
- SET LOCAL
- ALTER TABLE APPEND
- The `registerOutParameter` method call through the Java Database Connectivity (JDBC) driver isn't supported for the `refcursor` data type. For an example of using the `refcursor` data type, see [Returning a result set \(p. 168\)](#).
- The NONATOMIC transaction mode is not supported when the stored procedure is created using JDBC or ODBC drivers.

PL/pgSQL language reference

Stored procedures in Amazon Redshift are based on the PostgreSQL PL/pgSQL procedural language, with some important differences. In this reference, you can find details of PL/pgSQL syntax as implemented by Amazon Redshift. For more information about PL/pgSQL, see [PL/pgSQL - SQL procedural language](#) in the PostgreSQL documentation.

Topics

- [PL/pgSQL reference conventions \(p. 182\)](#)
- [Structure of PL/pgSQL \(p. 182\)](#)
- [Supported PL/pgSQL statements \(p. 186\)](#)

PL/pgSQL reference conventions

In this section, you can find the conventions that are used to write the syntax for the PL/pgSQL stored procedure language.

Character	Description
CAPS	Words in capital letters are keywords.
[]	Brackets denote optional arguments. Multiple arguments in brackets indicate that you can choose any number of the arguments. In addition, arguments in brackets on separate lines indicate that the Amazon Redshift parser expects the arguments to be in the order that they are listed in the syntax.
{ }	Braces indicate that you are required to choose one of the arguments inside the braces.
	Pipes indicate that you can choose between the arguments.
<i>red italics</i>	Words in red italics indicate placeholders. Insert the appropriate value in place of the word in red italics.
...	An ellipsis indicates that you can repeat the preceding element.
'	Words in single quotation marks indicate that you must type the quotes.

Structure of PL/pgSQL

PL/pgSQL is a procedural language with many of the same constructs as other procedural languages.

Topics

- [Block \(p. 183\)](#)
- [Variable declaration \(p. 184\)](#)
- [Alias declaration \(p. 184\)](#)
- [Built-in variables \(p. 184\)](#)
- [Record types \(p. 186\)](#)

Block

PL/pgSQL is a block-structured language. The complete body of a procedure is defined in a block, which contains variable declarations and PL/pgSQL statements. A statement can also be a nested block, or subblock.

End declarations and statements with a semicolon. Follow the END keyword in a block or subblock with a semicolon. Don't use semicolons after the keywords DECLARE and BEGIN.

You can write all keywords and identifiers in mixed uppercase and lowercase. Identifiers are implicitly converted to lowercase unless enclosed in double quotation marks.

A double hyphen (--) starts a comment that extends to the end of the line. A /* starts a block comment that extends to the next occurrence of */. You can't nest block comments. However, you can enclose double-hyphen comments in a block comment, and a double hyphen can hide the block comment delimiters /* and */.

Any statement in the statement section of a block can be a subblock. You can use subblocks for logical grouping or to localize variables to a small group of statements.

```
[ <>label>> ]
[ DECLARE
    declarations ]
BEGIN
    statements
END [ label ];
```

The variables declared in the declarations section preceding a block are initialized to their default values every time the block is entered. In other words, they're not initialized only once per function call.

The following shows an example.

```
CREATE PROCEDURE update_value() AS $$
DECLARE
    value integer := 20;
BEGIN
    RAISE NOTICE 'Value here is %', value; -- Value here is 20
    value := 50;
    --
    -- Create a subblock
    --
    DECLARE
        value integer := 80;
    BEGIN
        RAISE NOTICE 'Value here is %', value; -- Value here is 80
    END;

    RAISE NOTICE 'Value here is %', value; -- Value here is 50
END;
$$ LANGUAGE plpgsql;
```

Use a label to identify the block to use in an EXIT statement or to qualify the names of the variables declared in the block.

Don't confuse the use of BEGIN/END for grouping statements in PL/pgSQL with the database commands for transaction control. The BEGIN and END in PL/pgSQL are only for grouping. They don't start or end a transaction.

Variable declaration

Declare all variables in a block, except for loop variables, in the block's DECLARE section. Variables can use any valid Amazon Redshift data type. For supported data types, see [Data types \(p. 574\)](#).

PL/pgSQL variables can be any Amazon Redshift supported data type, plus RECORD and refcursor. For more information about RECORD, see [Record types \(p. 186\)](#). For more information about refcursor, see [Cursors \(p. 195\)](#).

```
DECLARE
name [ CONSTANT ] type [ NOT NULL ] [ { DEFAULT | := } expression ];
```

Following, you can find example variable declarations.

```
customerID integer;
numberofitems numeric(6);
link varchar;
onerow RECORD;
```

The loop variable of a FOR loop iterating over a range of integers is automatically declared as an integer variable.

The DEFAULT clause, if given, specifies the initial value assigned to the variable when the block is entered. If the DEFAULT clause is not given, then the variable is initialized to the SQL NULL value. The CONSTANT option prevents the variable from being assigned to, so that its value remains constant for the duration of the block. If NOT NULL is specified, an assignment of a null value results in a runtime error. All variables declared as NOT NULL must have a non-null default value specified.

The default value is evaluated every time the block is entered. For example, assigning now() to a variable of type timestamp causes the variable to have the time of the current function call, not the time when the function was precompiled.

```
quantity INTEGER DEFAULT 32;
url VARCHAR := 'http://mysite.com';
user_id CONSTANT INTEGER := 10;
```

The refcursor data type is the data type of cursor variables within stored procedures. A refcursor value can be returned from within a stored procedure. For more information, see [Returning a result set \(p. 168\)](#).

Alias declaration

If stored procedure's signature omits the argument name, you can declare an alias for the argument.

```
name ALIAS FOR $n;
```

Built-in variables

The following built-in variables are supported:

- FOUND
- SQLSTATE
- SQLERRM
- GET DIAGNOSTICS integer_var := ROW_COUNT;

FOUND is a special variable of type Boolean. FOUND starts out false within each procedure call. FOUND is set by the following types of statements:

- SELECT INTO

Sets FOUND to true if it returns a row, false if no row is returned.

- UPDATE, INSERT, and DELETE

Sets FOUND to true if at least one row is affected, false if no row is affected.

- FETCH

Sets FOUND to true if it returns a row, false if no row is returned.

- FOR statement

Sets FOUND to true if the FOR statement iterates one or more times, and otherwise false. This applies to all three variants of the FOR statement: integer FOR loops, record-set FOR loops, and dynamic record-set FOR loops.

FOUND is set when the FOR loop exits. Inside the runtime of the loop, FOUND isn't modified by the FOR statement. However, it can be changed by running other statements within the loop body.

The following shows an example.

```
CREATE TABLE employee(empname varchar);
CREATE OR REPLACE PROCEDURE show_found()
AS $$ 
DECLARE
    myrec record;
BEGIN
    SELECT INTO myrec * FROM employee WHERE empname = 'John';
    IF NOT FOUND THEN
        RAISE EXCEPTION 'employee John not found';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Within an exception handler, the special variable SQLSTATE contains the error code that corresponds to the exception that was raised. The special variable SQLERRM contains the error message associated with the exception. These variables are undefined outside exception handlers and display an error if used.

The following shows an example.

```
CREATE OR REPLACE PROCEDURE sqlstate_sqlerrm() AS
$$
BEGIN
    UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
    EXECUTE 'select invalid';
    EXCEPTION WHEN OTHERS THEN
        RAISE INFO 'error message SQLERRM %', SQLERRM;
        RAISE INFO 'error message SQLSTATE %', SQLSTATE;
END;
$$ LANGUAGE plpgsql;
```

ROW_COUNT is used with the GET DIAGNOSTICS command. It shows the number of rows processed by the last SQL command sent down to the SQL engine.

The following shows an example.

```
CREATE OR REPLACE PROCEDURE sp_row_count() AS
$$
DECLARE
    integer_var int;
BEGIN
    INSERT INTO tbl_row_count VALUES(1);
    GET DIAGNOSTICS integer_var := ROW_COUNT;
    RAISE INFO 'rows inserted = %', integer_var;
END;
$$ LANGUAGE plpgsql;
```

Record types

A RECORD type is not a true data type, only a placeholder. Record type variables assume the actual row structure of the row that they are assigned during a SELECT or FOR command. The substructure of a record variable can change each time it is assigned a value. Until a record variable is first assigned to, it has no substructure. Any attempt to access a field in it throws a runtime error.

```
name RECORD;
```

The following shows an example.

```
CREATE TABLE tbl_record(a int, b int);
INSERT INTO tbl_record VALUES(1, 2);
CREATE OR REPLACE PROCEDURE record_example()
LANGUAGE plpgsql
AS $$
DECLARE
    rec RECORD;
BEGIN
    FOR rec IN SELECT a FROM tbl_record
    LOOP
        RAISE INFO 'a = %', rec.a;
    END LOOP;
END;
$$;
```

Supported PL/pgSQL statements

PL/pgSQL statements augment SQL commands with procedural constructs, including looping and conditional expressions, to control logical flow. Most SQL commands can be used, including data manipulation language (DML) such as COPY, UNLOAD, and INSERT, and data definition language (DDL) such as CREATE TABLE. For a list of comprehensive SQL commands, see [SQL commands \(p. 623\)](#). In addition, the following PL/pgSQL statements are supported by Amazon Redshift.

Topics

- [Assignment \(p. 187\)](#)
- [SELECT INTO \(p. 187\)](#)
- [No-op \(p. 188\)](#)
- [Dynamic SQL \(p. 188\)](#)
- [Return \(p. 189\)](#)

- [Conditionals: IF \(p. 189\)](#)
- [Conditionals: CASE \(p. 190\)](#)
- [Loops \(p. 191\)](#)
- [Cursors \(p. 195\)](#)
- [RAISE \(p. 196\)](#)
- [Transaction control \(p. 197\)](#)

Assignment

The assignment statement assigns a value to a variable. The expression must return a single value.

```
identifier := expression;
```

Using the nonstandard = for assignment, instead of :=, is also accepted.

If the data type of the expression doesn't match the variable's data type or the variable has a size or precision, the result value is implicitly converted.

The following shows examples.

```
customer_number := 20;
tip := subtotal * 0.15;
```

SELECT INTO

The SELECT INTO statement assigns the result of multiple columns (but only one row) into a record variable or list of scalar variables.

```
SELECT INTO target select_expressions FROM ...;
```

In the preceding syntax, *target* can be a record variable or a comma-separated list of simple variables and record fields. The *select_expressions* list and the remainder of the command are the same as in regular SQL.

If a variable list is used as *target*, the selected values must exactly match the structure of the target, or a runtime error occurs. When a record variable is the target, it automatically configures itself to the row type of the query result columns.

The INTO clause can appear almost anywhere in the SELECT statement. It usually appears just after the SELECT clause, or just before FROM clause. That is, it appears just before or just after the *select_expressions* list.

If the query returns zero rows, NULL values are assigned to *target*. If the query returns multiple rows, the first row is assigned to *target* and the rest are discarded. Unless the statement contains an ORDER BY, the first row is not deterministic.

To determine whether the assignment returned at least one row, use the special FOUND variable.

```
SELECT INTO customer_rec * FROM cust WHERE custname = lname;
IF NOT FOUND THEN
    RAISE EXCEPTION 'employee % not found', lname;
END IF;
```

To test whether a record result is null, you can use the IS NULL conditional. There is no way to determine whether any additional rows might have been discarded. The following example handles the case where no rows have been returned.

```
CREATE OR REPLACE PROCEDURE select_into_null(return_webpage OUT varchar(256))
AS $$ 
DECLARE
    customer_rec RECORD;
BEGIN
    SELECT INTO customer_rec * FROM users WHERE user_id=3;
    IF customer_rec.webpage IS NULL THEN
        -- user entered no webpage, return "http://"
        return_webpage = 'http://';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

No-op

The no-op statement (NULL;) is a placeholder statement that does nothing. A no-op statement can indicate that one branch of an IF-THEN-ELSE chain is empty.

```
NULL;
```

Dynamic SQL

To generate dynamic commands that can involve different tables or different data types each time they are run from a PL/pgSQL stored procedure, use the EXECUTE statement.

```
EXECUTE command-string [ INTO target ];
```

In the preceding, *command-string* is an expression yielding a string (of type text) that contains the command to be run. This *command-string* value is sent to the SQL engine. No substitution of PL/pgSQL variables is done on the command string. The values of variables must be inserted in the command string as it is constructed.

Note

You can't use COMMIT and ROLLBACK statements from within dynamic SQL. For information about using COMMIT and ROLLBACK statements within a stored procedure, see [Managing transactions \(p. 169\)](#).

When working with dynamic commands, you often have to handle escaping of single quotation marks. We recommend enclosing fixed text in quotation marks in your function body using dollar quoting. Dynamic values to insert into a constructed query require special handling because they might themselves contain quotation marks. The following example assumes dollar quoting for the function as a whole, so the quotation marks don't need to be doubled.

```
EXECUTE 'UPDATE tbl SET '
|| quote_ident(colname)
|| ' = '
|| quote_literal(newvalue)
|| ' WHERE key = '
|| quote_literal(keyvalue);
```

The preceding example shows the functions `quote_ident(text)` and `quote_literal(text)`. This example passes variables that contain column and table identifiers to the `quote_ident` function. It also passes variables that contain literal strings in the constructed command to the `quote_literal`

function. Both functions take the appropriate steps to return the input text enclosed in double or single quotation marks respectively, with any embedded special characters properly escaped.

Dollar quoting is only useful for quoting fixed text. Don't write the preceding example in the following format.

```
EXECUTE 'UPDATE tbl SET '
|| quote_ident(colname)
|| ' = $$'
|| newvalue
|| '$$ WHERE key = '
|| quote_literal(keyvalue);
```

You don't do this because the example breaks if the contents of newvalue happen to contain `$$`. The same problem applies to any other dollar-quoting delimiter that you might choose. To safely quote text that is not known in advance, use the `quote_literal` function.

Return

The `RETURN` statement returns back to the caller from a stored procedure.

```
RETURN;
```

The following shows an example.

```
CREATE OR REPLACE PROCEDURE return_example(a int)
AS $$ 
BEGIN
    FOR b in 1..10 LOOP
        IF b < a THEN
            RAISE INFO 'b = %', b;
        ELSE
            RETURN;
        END IF;
    END LOOP;
END;
$$ LANGUAGE plpgsql;
```

Conditionals: IF

The `IF` conditional statement can take the following forms in the PL/pgSQL language that Amazon Redshift uses:

- `IF ... THEN`

```
IF boolean-expression THEN
    statements
END IF;
```

The following shows an example.

```
IF v_user_id <> 0 THEN
    UPDATE users SET email = v_email WHERE user_id = v_user_id;
END IF;
```

- `IF ... THEN ... ELSE`

```
IF boolean-expression THEN
```

```

statements
ELSE
  statements
END IF;

```

The following shows an example.

```

IF parentid IS NULL OR parentid = ''
THEN
  return_name = fullname;
  RETURN;
ELSE
  return_name = hp_true_filename(parentid) || '/' || fullname;
  RETURN;
END IF;

```

- IF ... THEN ... ELSIF ... THEN ... ELSE

The key word ELSIF can also be spelled ELSEIF.

```

IF boolean-expression THEN
  statements
[ ELSIF boolean-expression THEN
  statements
[ ELSIF boolean-expression THEN
  statements
  ...] ]
[ ELSE
  statements ]
END IF;

```

The following shows an example.

```

IF number = 0 THEN
  result := 'zero';
ELSIF number > 0 THEN
  result := 'positive';
ELSIF number < 0 THEN
  result := 'negative';
ELSE
  -- the only other possibility is that number is null
  result := 'NULL';
END IF;

```

Conditionals: CASE

The CASE conditional statement can take the following forms in the PL/pgSQL language that Amazon Redshift uses:

- Simple CASE

```

CASE search-expression
WHEN expression [, expression [ ... ]] THEN
  statements
[ WHEN expression [, expression [ ... ]] THEN
  statements
  ...
]
[ ELSE
  statements ]

```

```
END CASE;
```

A simple CASE statement provides conditional execution based on equality of operands.

The *search-expression* value is evaluated one time and successively compared to each *expression* in the WHEN clauses. If a match is found, then the corresponding *statements* run, and then control passes to the next statement after END CASE. Subsequent WHEN expressions aren't evaluated. If no match is found, the ELSE *statements* run. However, if ELSE isn't present, then a CASE_NOT_FOUND exception is raised.

The following shows an example.

```
CASE x
WHEN 1, 2 THEN
    msg := 'one or two';
ELSE
    msg := 'other value than one or two';
END CASE;
```

- Searched CASE

```
CASE
WHEN boolean-expression THEN
    statements
[ WHEN boolean-expression THEN
    statements
    ...
]
[ ELSE
    statements ]
END CASE;
```

The searched form of CASE provides conditional execution based on truth of Boolean expressions.

Each WHEN clause's *boolean-expression* is evaluated in turn, until one is found that yields true. Then the corresponding statements run, and then control passes to the next statement after END CASE. Subsequent WHEN *expressions* aren't evaluated. If no true result is found, the ELSE *statements* are run. However, if ELSE isn't present, then a CASE_NOT_FOUND exception is raised.

The following shows an example.

```
CASE
WHEN x BETWEEN 0 AND 10 THEN
    msg := 'value is between zero and ten';
WHEN x BETWEEN 11 AND 20 THEN
    msg := 'value is between eleven and twenty';
END CASE;
```

Loops

Loop statements can take the following forms in the PL/pgSQL language that Amazon Redshift uses:

- Simple loop

```
[<<label>>]
LOOP
    statements
END LOOP [ label ];
```

A simple loop defines an unconditional loop that is repeated indefinitely until terminated by an EXIT or RETURN statement. The optional label can be used by EXIT and CONTINUE statements within nested loops to specify which loop the EXIT and CONTINUE statements refer to.

The following shows an example.

```
CREATE OR REPLACE PROCEDURE simple_loop()
LANGUAGE plpgsql
AS $$
BEGIN
    <<simple_while>>
    LOOP
        RAISE INFO 'I am raised once';
        EXIT simple_while;
        RAISE INFO 'I am not raised';
    END LOOP;
    RAISE INFO 'I am raised once as well';
END;
$$;
```

- Exit loop

```
EXIT [ label ] [ WHEN expression ];
```

If *label* isn't present, the innermost loop is terminated and the statement following the END LOOP runs next. If *label* is present, it must be the label of the current or some outer level of nested loop or block. Then, the named loop or block is terminated and control continues with the statement after the loop or block corresponding END.

If WHEN is specified, the loop exit occurs only if *expression* is true. Otherwise, control passes to the statement after EXIT.

You can use EXIT with all types of loops; it isn't limited to use with unconditional loops.

When used with a BEGIN block, EXIT passes control to the next statement after the end of the block. A label must be used for this purpose. An unlabeled EXIT is never considered to match a BEGIN block.

The following shows an example.

```
CREATE OR REPLACE PROCEDURE simple_loop_when(x int)
LANGUAGE plpgsql
AS $$
DECLARE i INTEGER := 0;
BEGIN
    <<simple_loop_when>>
    LOOP
        RAISE INFO 'i %', i;
        i := i + 1;
        EXIT simple_loop_when WHEN (i >= x);
    END LOOP;
END;
$$;
```

- Continue loop

```
CONTINUE [ label ] [ WHEN expression ];
```

If *label* is not given, the execution jumps to the next iteration of the innermost loop. That is, all statements remaining in the loop body are skipped. Control then returns to the loop control

expression (if any) to determine whether another loop iteration is needed. If *label* is present, it specifies the label of the loop whose execution is continued.

If WHEN is specified, the next iteration of the loop is begun only if *expression* is true. Otherwise, control passes to the statement after CONTINUE.

You can use CONTINUE with all types of loops; it isn't limited to use with unconditional loops.

```
CONTINUE mylabel;
```

- WHILE loop

```
[<<label>>]
WHILE expression LOOP
    statements
END LOOP [ label ];
```

The WHILE statement repeats a sequence of statements so long as the *boolean-expression* evaluates to true. The expression is checked just before each entry to the loop body.

The following shows an example.

```
WHILE amount_owed > 0 AND gift_certificate_balance > 0 LOOP
    -- some computations here
END LOOP;

WHILE NOT done LOOP
    -- some computations here
END LOOP;
```

- FOR loop (integer variant)

```
[<<label>>]
FOR name IN [ REVERSE ] expression .. expression LOOP
    statements
END LOOP [ label ];
```

The FOR loop (integer variant) creates a loop that iterates over a range of integer values. The variable name is automatically defined as type integer and exists only inside the loop. Any existing definition of the variable name is ignored within the loop. The two expressions giving the lower and upper bound of the range are evaluated one time when entering the loop. If you specify REVERSE, then the step value is subtracted, rather than added, after each iteration.

If the lower bound is greater than the upper bound (or less than, in the REVERSE case), the loop body doesn't run. No error is raised.

If a label is attached to the FOR loop, then you can reference the integer loop variable with a qualified name, using that label.

The following shows an example.

```
FOR i IN 1..10 LOOP
    -- i will take on the values 1,2,3,4,5,6,7,8,9,10 within the loop
END LOOP;

FOR i IN REVERSE 10..1 LOOP
    -- i will take on the values 10,9,8,7,6,5,4,3,2,1 within the loop
END LOOP;
```

- FOR loop (result set variant)

```
[<<label>>]
FOR target IN query LOOP
    statements
END LOOP [ label ];
```

The **target** is a record variable or comma-separated list of scalar variables. The target is successively assigned each row resulting from the query, and the loop body is run for each row.

The FOR loop (result set variant) enables a stored procedure to iterate through the results of a query and manipulate that data accordingly.

The following shows an example.

```
CREATE PROCEDURE cs_refresh_reports() AS $$
DECLARE
    reports RECORD;
BEGIN
    FOR reports IN SELECT * FROM cs_reports ORDER BY sort_key LOOP
        -- Now "reports" has one record from cs_reports
        EXECUTE 'TRUNCATE TABLE ' || quote_ident(reports.report_name);
        EXECUTE 'INSERT INTO ' || quote_ident(reports.report_name) || ' ' || reports.report_query;
    END LOOP;
    RETURN;
END;
$$ LANGUAGE plpgsql;
```

- FOR loop with dynamic SQL

```
[<<label>>]
FOR record_or_row IN EXECUTE text_expression LOOP
    statements
END LOOP;
```

A FOR loop with dynamic SQL enables a stored procedure to iterate through the results of a dynamic query and manipulate that data accordingly.

The following shows an example.

```
CREATE OR REPLACE PROCEDURE for_loop_dynamic_sql(x int)
LANGUAGE plpgsql
AS $$
DECLARE
    rec RECORD;
    query text;
BEGIN
    query := 'SELECT * FROM tbl_dynamic_sql LIMIT ' || x;
    FOR rec IN EXECUTE query
    LOOP
        RAISE INFO 'a %', rec.a;
    END LOOP;
END;
$$;
```

Cursors

Rather than running a whole query at once, you can set up a cursor. A *cursor* encapsulates a query and reads the query result a few rows at a time. One reason for doing this is to avoid memory overrun when the result contains a large number of rows. Another reason is to return a reference to a cursor that a stored procedure has created, which allows the caller to read the rows. This approach provides an efficient way to return large row sets from stored procedures.

To use cursors in a NONATOMIC stored procedure, place the cursor loop between START TRANSACTION...COMMIT.

To set up a cursor, first you declare a cursor variable. All access to cursors in PL/pgSQL goes through cursor variables, which are always of the special data type `refcursor`. A `refcursor` data type simply holds a reference to a cursor.

You can create a cursor variable by declaring it as a variable of type `refcursor`. Or, you can use the cursor declaration syntax following.

```
name CURSOR [ ( arguments ) ] FOR query ;
```

In the preceding, *arguments* (if specified) is a comma-separated list of *name datatype* pairs that each define names to be replaced by parameter values in *query*. The actual values to substitute for these names are specified later, when the cursor is opened.

The following shows examples.

```
DECLARE
  curs1 refcursor;
  curs2 CURSOR FOR SELECT * FROM tenk1;
  curs3 CURSOR (key integer) IS SELECT * FROM tenk1 WHERE unique1 = key;
```

All three of these variables have the data type `refcursor`, but the first can be used with any query. In contrast, the second has a fully specified query already bound to it, and the last has a parameterized query bound to it. The key value is replaced by an integer parameter value when the cursor is opened. The variable `curs1` is said to be *unbound* because it is not bound to any particular query.

Before you can use a cursor to retrieve rows, it must be opened. PL/pgSQL has three forms of the OPEN statement, of which two use unbound cursor variables and the third uses a bound cursor variable:

- **Open for select:** The cursor variable is opened and given the specified query to run. The cursor can't be open already. Also, it must have been declared as an unbound cursor (that is, as a simple `refcursor` variable). The `SELECT` query is treated in the same way as other `SELECT` statements in PL/pgSQL.

```
OPEN cursor_name FOR SELECT ...;
```

The following shows an example.

```
OPEN curs1 FOR SELECT * FROM foo WHERE key = mykey;
```

- **Open for execute:** The cursor variable is opened and given the specified query to run. The cursor can't be open already. Also, it must have been declared as an unbound cursor (that is, as a simple `refcursor` variable). The query is specified as a string expression in the same way as in the EXECUTE command. This approach gives flexibility so the query can vary from one run to the next.

```
OPEN cursor_name FOR EXECUTE query_string;
```

The following shows an example.

```
OPEN curs1 FOR EXECUTE 'SELECT * FROM ' || quote_ident($1);
```

- Open a bound cursor: This form of OPEN is used to open a cursor variable whose query was bound to it when it was declared. The cursor can't be open already. A list of actual argument value expressions must appear if and only if the cursor was declared to take arguments. These values are substituted in the query.

```
OPEN bound_cursor_name [ ( argument_values ) ];
```

The following shows an example.

```
OPEN curs2;  
OPEN curs3(42);
```

After a cursor has been opened, you can work with it by using the statements described following. These statements don't have to occur in the same stored procedure that opened the cursor. You can return a refcursor value out of a stored procedure and let the caller operate on the cursor. All portals are implicitly closed at transaction end. Thus, you can use a refcursor value to reference an open cursor only until the end of the transaction.

- FETCH retrieves the next row from the cursor into a target. This target can be a row variable, a record variable, or a comma-separated list of simple variables, just as with SELECT INTO. As with SELECT INTO, you can check the special variable FOUND to see whether a row was obtained.

```
FETCH cursor INTO target;
```

The following shows an example.

```
FETCH curs1 INTO rowvar;
```

- CLOSE closes the portal underlying an open cursor. You can use this statement to release resources earlier than end of the transaction. You can also use this statement to free the cursor variable to be opened again.

```
CLOSE cursor;
```

The following shows an example.

```
CLOSE curs1;
```

RAISE

Use the RAISE level statement to report messages and raise errors.

```
RAISE level 'format' [, variable [, ...]];
```

Possible levels are NOTICE, INFO, LOG, WARNING, and EXCEPTION. EXCEPTION raises an error, which normally cancels the current transaction. The other levels generate only messages of different priority levels.

Inside the format string, % is replaced by the next optional argument's string representation. Write % to emit a literal %. Currently, optional arguments must be simple variables, not expressions, and the format must be a simple string literal.

In the following example, the value of v_job_id replaces the % in the string.

```
RAISE NOTICE 'Calling cs_create_job(%)', v_job_id;
```

Use the RAISE statement to re-throw the exception caught by an exception handling block. This statement is only valid in exception handling blocks of NONATOMIC mode stored procedures.

```
RAISE;
```

Transaction control

You can work with transaction control statements in the PL/pgSQL language that Amazon Redshift uses. For information about using the statements COMMIT, ROLLBACK, and TRUNCATE within a stored procedure, see [Managing transactions \(p. 169\)](#).

In NONATOMIC mode stored procedures, use START TRANSACTION to start a transaction block.

```
START TRANSACTION;
```

Note

The PL/pgSQL statement START TRANSACTION is different from the SQL command START TRANSACTION in the following ways:

- Within stored procedures, START TRANSACTION is not synonymous with BEGIN.
- The PL/pgSQL statement does not support optional isolation level and access permission keywords.

Creating materialized views in Amazon Redshift

In a data warehouse environment, applications often must perform complex queries on large tables. An example is SELECT statements that perform multi-table joins and aggregations on tables that contain billions of rows. Processing these queries can be expensive, in terms of system resources and the time it takes to compute the results.

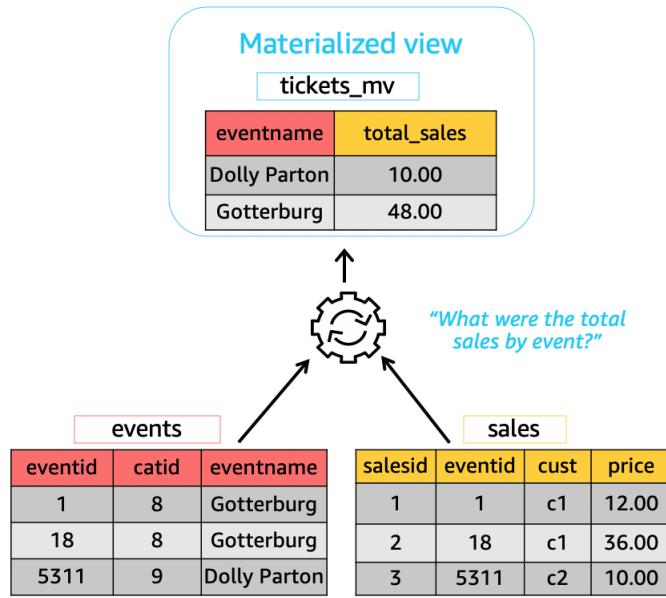
Materialized views in Amazon Redshift provide a way to address these issues. A *materialized view* contains a precomputed result set, based on an SQL query over one or more base tables. You can issue SELECT statements to query a materialized view, in the same way that you can query other tables or views in the database. Amazon Redshift returns the precomputed results from the materialized view, without having to access the base tables at all. From the user standpoint, the query results are returned much faster compared to when retrieving the same data from the base tables.

Materialized views are especially useful for speeding up queries that are predictable and repeated. Instead of performing resource-intensive queries against large tables (such as aggregates or multiple joins), applications can query a materialized view and retrieve a precomputed result set. For example, consider the scenario where a set of queries is used to populate dashboards, such as Amazon QuickSight. This use case is ideal for a materialized view, because the queries are predictable and repeated over and over again.

You can define a materialized view in terms of other materialized views. Use *materialized views on materialized views* to expand the capability of materialized views. In this approach, an existing materialized view plays the same role as a base table for the query to retrieve data.

This approach is especially useful for reusing precomputed joins for different aggregate or GROUP BY options. For example, take a materialized view that joins customer information (containing millions of rows) with item order detail information (containing billions of rows). This is an expensive query to compute on demand repeatedly. You can use different GROUP BY options for the materialized views created on top of this materialized view and join with other tables. Doing this saves compute time otherwise used to run the expensive underlying join every time. The [STV_MV_DEPS \(p. 1587\)](#) table shows the dependencies of a materialized view on other materialized views.

When you create a materialized view, Amazon Redshift runs the user-specified SQL statement to gather the data from the base table or tables and stores the result set. The following illustration provides an overview of the materialized view `tickets_mv` that an SQL query defines by using two base tables, `events` and `sales`.



You can then use these materialized views in queries to speed them up. In addition, Amazon Redshift can automatically rewrite these queries to use materialized views, even when the query doesn't explicitly reference a materialized view. Automatic rewrite of queries is especially powerful in enhancing performance when you can't change your queries to use materialized views.

To update the data in the materialized view, you can use the `REFRESH MATERIALIZED VIEW` statement at any time to manually refresh materialized views. Amazon Redshift identifies changes that have taken place in the base table or tables, and then applies those changes to the materialized view. Because automatic rewriting of queries requires materialized views to be up to date, as a materialized view owner, make sure to refresh materialized views whenever a base table changes.

Amazon Redshift provides a few ways to keep materialized views up to date for automatic rewriting. You can configure materialized views with the automatic refresh option to refresh materialized views when base tables of materialized views are updated. This autorefresh operation runs at a time when cluster resources are available to minimize disruptions to other workloads. Because the scheduling of autorefresh is workload-dependent, you can have more control over when Amazon Redshift refreshes your materialized views. You can schedule a materialized view refresh job by using Amazon Redshift scheduler API and console integration. For more information about query scheduling, see [Scheduling a query on the Amazon Redshift console](#).

Doing this is especially useful when there is a service level agreement (SLA) requirement for up-to-date data from a materialized view. You can also manually refresh any materialized views that you can autorefresh. For information on how to create materialized views, see [CREATE MATERIALIZED VIEW \(p. 796\)](#).

You can issue `SELECT` statements to query a materialized view. For information on how to query materialized views, see [Querying a materialized view \(p. 200\)](#). The result set eventually becomes stale when data is inserted, updated, and deleted in the base tables. You can refresh the materialized view at any time to update it with the latest changes from the base tables. For information on how to refresh materialized views, see [REFRESH MATERIALIZED VIEW \(p. 914\)](#).

For details about SQL commands used to create and manage materialized views, see the following command topics:

- [CREATE MATERIALIZED VIEW \(p. 796\)](#)
- [ALTER MATERIALIZED VIEW \(p. 636\)](#)

- [REFRESH MATERIALIZED VIEW \(p. 914\)](#)
- [DROP MATERIALIZED VIEW \(p. 869\)](#)

For information about system tables and views to monitor materialized views, see the following topics:

- [STV_MV_INFO \(p. 1588\)](#)
- [STL_MV_STATE \(p. 1532\)](#)
- [SVL_MV_REFRESH_STATUS \(p. 1638\)](#)
- [STV_MV_DEPS \(p. 1587\)](#)

Topics

- [Querying a materialized view \(p. 200\)](#)
- [Automatic query rewriting to use materialized views \(p. 201\)](#)
- [Refreshing a materialized view \(p. 202\)](#)
- [Automated materialized views \(p. 204\)](#)
- [Using a user-defined function \(UDF\) in a materialized view \(p. 206\)](#)
- [Streaming ingestion \(p. 208\)](#)

Querying a materialized view

You can use a materialized view in any SQL query by referencing the materialized view name as the data source, like a table or standard view.

When a query accesses a materialized view, it sees only the data that is stored in the materialized view as of its most recent refresh. Thus, the query might not see all the latest changes from corresponding base tables of the materialized view.

If other users want to query the materialized view, the owner of the materialized view grants the SELECT permission to those users. The other users don't need to have the SELECT permission on the underlying base tables. The owner of the materialized view can also revoke the SELECT permission from other users to prevent them from querying the materialized view.

If the owner of the materialized view no longer has the SELECT permission on the underlying base tables:

- The owner can no longer query the materialized view.
- Other users who have the SELECT permission on the materialized view can no longer query the materialized view.

The following example queries the `tickets_mv` materialized view. For more information on the SQL command used to create a materialized view, see [CREATE MATERIALIZED VIEW \(p. 796\)](#).

```
SELECT sold
FROM tickets_mv
WHERE catgroup = 'Concerts';
```

Because the query results are precomputed, there's no need to access the underlying tables (`category`, `event`, and `sales`). Amazon Redshift can return the results directly from `tickets_mv`.

Automatic query rewriting to use materialized views

You can use automatic query rewriting of materialized views in Amazon Redshift to have Amazon Redshift rewrite queries to use materialized views. Doing this accelerates query workloads even for queries that don't explicitly reference a materialized view. When Amazon Redshift rewrites queries, it only uses materialized views that are up to date.

Usage notes

To check if automatic rewriting of queries is used for a query, you can inspect the query plan or `STL_EXPLAIN`. The following shows a `SELECT` statement and the `EXPLAIN` output of the original query plan.

```
SELECT catgroup, SUM(qtysold) AS sold
FROM category c, event e, sales s
WHERE c.catid = e.catid AND e.eventid = s.eventid
GROUP BY 1;

EXPLAIN
XN HashAggregate (cost=920021.24..920021.24 rows=1 width=35)
-> XN Hash Join DS_BCAST_INNER (cost=440004.53..920021.22 rows=4 width=35)
    Hash Cond: ("outer".eventid = "inner".eventid)
    -> XN Seq Scan on sales s (cost=0.00..7.40 rows=740 width=6)
    -> XN Hash (cost=440004.52..440004.52 rows=1 width=37)
        -> XN Hash Join DS_BCAST_INNER (cost=0.01..440004.52 rows=1 width=37)
            Hash Cond: ("outer".catid = "inner".catid)
            -> XN Seq Scan on event e (cost=0.00..2.00 rows=200 width=6)
            -> XN Hash (cost=0.01..0.01 rows=1 width=35)
                -> XN Seq Scan on category c (cost=0.00..0.01 rows=1 width=35)
```

The following shows the `EXPLAIN` output after a successful automatic rewriting. This output includes a scan on the materialized view in the query plan that replaces parts of the original query plan.

```
* EXPLAIN
  XN HashAggregate (cost=11.85..12.35 rows=200 width=41)
    -> XN Seq Scan on mv_tbl_tickets_mv_0 derived_table1 (cost=0.00..7.90 rows=790
width=41)
```

Only up-to-date (fresh) materialized views are considered for automatic rewriting of queries, irrespective of the refresh strategy, such as auto, scheduled, or manual. Hence, the original query returns up-to-date results. When a materialized view is explicitly referenced in queries, Amazon Redshift accesses currently stored data in the materialized view. This data might not reflect the latest changes from the base tables of the materialized view.

You can use automatic query rewriting of materialized views that are created on cluster version 1.0.20949 or later.

You can stop automatic query rewriting at the session level by using `SET mv_enable_aqmv_for_session` to `FALSE`.

Limitations

Following are limitations for using automatic query rewriting of materialized views:

- Automatic query rewriting works with materialized views that don't reference or include any of the following:
 - Subqueries
 - Left, right, or full outer joins
 - Set operations
 - Any aggregate functions, except SUM, COUNT, MIN, MAX, and AVG. (These are the only aggregate functions that work with automatic query rewriting.)
 - Any aggregate functions with DISTINCT
 - Any window functions
 - SELECT DISTINCT or HAVING clauses
 - External tables
 - Other materialized views
- Automatic query rewriting rewrites SELECT queries that refer to user-defined Amazon Redshift tables. Amazon Redshift doesn't rewrite the following queries:
 - CREATE TABLE AS statements
 - SELECT INTO statements
 - Queries on catalogs or system tables
 - Queries with outer joins or a SELECT DISTINCT clause
- If a query isn't automatically rewritten, check whether you have the SELECT permission on the specified materialized view and the [mv_enable_aqmv_for_session \(p. 1694\)](#) option is set to TRUE.

You can also check if your materialized views are eligible for automatic rewriting of queries by inspecting STV_MV_INFO. For more information, see [STV_MV_INFO \(p. 1588\)](#).

Refreshing a materialized view

When you create a materialized view, its contents reflect the state of the underlying database table or tables at that time. The data in the materialized view remains unchanged, even when applications change the data in the underlying tables. To update the data in the materialized view, you can use the REFRESH MATERIALIZED VIEW statement at any time to manually refresh materialized views. When you use this statement, Amazon Redshift identifies changes that have taken place in the base table or tables and applies those changes to the materialized view.

Amazon Redshift has two strategies for refreshing a materialized view:

- In many cases, Amazon Redshift can perform an incremental refresh. In an *incremental refresh*, Amazon Redshift quickly identifies the changes to the data in the base tables since the last refresh and updates the data in the materialized view. Incremental refresh is supported on the following SQL constructs used in the query when defining the materialized view:
 - Constructs that contain the clauses SELECT, FROM, [INNER] JOIN, WHERE, GROUP BY, or HAVING.
 - Constructs that contain aggregations, such as SUM, MIN, MAX, AVG, and COUNT.
 - Most built-in SQL functions, specifically the ones that are immutable, given that these have the same input arguments and always produce the same output.
- If an incremental refresh isn't possible, then Amazon Redshift performs a full refresh. A *full refresh* reruns the underlying SQL statement, replacing all of the data in the materialized view.
- Amazon Redshift automatically chooses the refresh method for a materialized view depending on the SELECT query used to define the materialized view.

Refreshing a materialized view on a materialized view isn't a cascading process. In other words, suppose that you have a materialized view A that depends on materialized view B. In this case, when the REFRESH

MATERIALIZED VIEW A is invoked, A is refreshed using the current version of B, even when B is out-of-date. To bring A fully up to date, before refreshing A, first refresh B in a separate transaction.

The following example shows how you can create a full refresh plan for a materialized view programmatically. To refresh the materialized view v, first refresh materialized view u. To refresh materialized view w, first refresh materialized view u and then materialized view v.

```

CREATE TABLE t(a INT);
CREATE MATERIALIZED VIEW u AS SELECT * FROM t;
CREATE MATERIALIZED VIEW v AS SELECT * FROM u;
CREATE MATERIALIZED VIEW w AS SELECT * FROM v;

WITH RECURSIVE recursive_deps (mv_tgt, lvl, mv_dep) AS
( SELECT trim(name) as mv_tgt, 0 as lvl, trim(ref_name) as mv_dep
  FROM stv_mv_deps
 UNION ALL
  SELECT R.mv_tgt, R.lvl+1 as lvl, trim(S.ref_name) as mv_dep
    FROM stv_mv_deps S, recursive_deps R
   WHERE R.mv_dep = S.name
)
SELECT mv_tgt, mv_dep from recursive_deps
ORDER BY mv_tgt, lvl DESC;

mv_tgt | mv_dep
-----+-----
v      | u
w      | u
w      | v
(3 rows)

```

The following example shows an informative message when you run REFRESH MATERIALIZED VIEW on a materialized view that depends on an out-of-date materialized view.

```

create table a(a int);

create materialized view b as select * from a;

create materialized view c as select * from b;

insert into a values (1);

refresh materialized view c;
INFO: Materialized view c is already up to date. However, it depends on another
materialized view that is not up to date.

REFRESH MATERIALIZED VIEW b;
INFO: Materialized view b was incrementally updated successfully.

REFRESH MATERIALIZED VIEW c;
INFO: Materialized view c was incrementally updated successfully.

```

Amazon Redshift currently has the following limitations for incremental refresh for materialized views.

Amazon Redshift doesn't support incremental refresh for materialized views that are defined with a query using the following SQL elements:

- OUTER JOIN (RIGHT, LEFT, or FULL).
- The set operations UNION, INTERSECT, EXCEPT, and MINUS.
- The aggregate functions MEDIAN, PERCENTILE_CONT, LISTAGG, STDDEV_SAMP, STDDEV_POP, APPROXIMATE COUNT, APPROXIMATE PERCENTILE, and bitwise aggregate functions.

Note

The COUNT, SUM, and AVG aggregate functions are supported.

- DISTINCT aggregate functions, such as DISTINCT COUNT, DISTINCT SUM, and so on.
- Window functions.
- A query that uses temporary tables for query optimization, such as optimizing common subexpressions.
- Subqueries in any place other than the FROM clause.
- External tables referenced as base tables in the query that defines the materialized view.

Autorefreshing a materialized view

Amazon Redshift can automatically refresh materialized views with up-to-date data from its base tables when materialized views are created with or altered to have the autorefresh option. Amazon Redshift autorefreshes materialized views as soon as possible after base tables changes.

To complete refresh of the most important materialized views with minimal impact to active workloads in your cluster, Amazon Redshift considers multiple factors. These factors include current system load, the resources needed for refresh, available cluster resources, and how often the materialized views are used.

Amazon Redshift prioritizes your workloads over autorefresh and might stop autorefresh to preserve the performance of user workload. This approach might delay refresh of some materialized views. In some cases, you might need more deterministic refresh behavior for your materialized views. If so, consider using manual refresh as described in [REFRESH MATERIALIZED VIEW \(p. 914\)](#) or scheduled refresh using the Amazon Redshift scheduler API operations or the console.

You can set autorefresh for materialized views using CREATE MATERIALIZED VIEW. You can also use the AUTO REFRESH clause to refresh materialized views automatically. For more information about creating materialized views, see [CREATE MATERIALIZED VIEW \(p. 796\)](#). You can turn on autorefresh for a current materialized view by using [ALTER MATERIALIZED VIEW \(p. 636\)](#).

Consider the following when you refresh materialized views:

- You can still refresh a materialized view explicitly using REFRESH MATERIALIZED VIEW command even if you haven't enabled autorefresh for the materialized view.
- Amazon Redshift doesn't autorefresh materialized views defined on external tables.
- For refresh status, you can check SVL_MV_REFRESH_STATUS, which records queries that were user-initiated or autorefreshed.
- To run REFRESH on recompute-only materialized views, make sure that you have the CREATE permission on schemas. For more information, see [GRANT \(p. 887\)](#).

Automated materialized views

Materialized views are a powerful tool for improving query performance in Amazon Redshift. They do this by storing a precomputed result set. Similar queries don't have to re-run the same logic each time, because they can retrieve records from the existing result set. Developers and analysts create materialized views after analyzing their workloads to determine which queries would benefit, and

whether the maintenance cost of each materialized view is worthwhile. As workloads grow or change, these materialized views must be reviewed to ensure they continue to provide tangible performance benefits.

The Automated Materialized Views (AutoMV) feature in Redshift provides the same performance benefits of user-created materialized views. Amazon Redshift continually monitors the workload using machine learning and creates new materialized views when they are beneficial. AutoMV balances the costs of creating and keeping materialized views up to date against expected benefits to query latency. The system also monitors previously created AutoMVs and drops them when they are no longer beneficial.

AutoMV behavior and capabilities are the same as user-created materialized views. They are refreshed automatically and incrementally, using the same criteria and restrictions. Just like materialized views created by users, [Automatic query rewriting to use materialized views \(p. 201\)](#) identifies queries that can benefit from system-created AutoMVs. It automatically rewrites those queries to use the AutoMVs, improving query performance. Developers don't need to revise queries to take advantage of AutoMV.

Note

Automated materialized views are refreshed intermittently. Queries rewritten to use AutoMV always return the latest results. When Redshift detects that data isn't up to date, queries aren't rewritten to read from automated materialized views. Instead, queries select the latest data from base tables.

Any workload with queries that are used repeatedly can benefit from AutoMV. Common use cases include:

- *Dashboards* - Dashboards are widely used to provide quick views of key business indicators (KPIs), events, trends, and other metrics. They often have a common layout with charts and tables, but show different views for filtering, or for dimension-selection operations, like drill down. Dashboards often have a common set of queries used repeatedly with different parameters. Dashboard queries can benefit greatly from automated materialized views.
- *Reports* - Reporting queries may be scheduled at various frequencies, based on business requirements and the type of report. Additionally, they can be automated or on-demand. A common characteristic of reporting queries is that they can be long running and resource-intensive. With AutoMV, these queries don't need to be recomputed each time they run, which reduces runtime for each query and resource utilization in Redshift.

To turn off automated materialized views, you update the `auto_mv` parameter group to `false`. For more information, see [Amazon Redshift parameter groups](#) in the Amazon Redshift Cluster Management Guide.

SQL scope and considerations for automated materialized views

- An automated materialized view can be initiated and created by a query or subquery, provided it contains a GROUP BY clause or one of the following aggregate functions: SUM, COUNT, MIN, MAX or AVG. But it cannot contain any of the following:
 - Left, right, or full outer joins
 - Aggregate functions other than SUM, COUNT, MIN, MAX, and AVG. (These particular functions work with automatic query rewriting.)
 - Any aggregate function that includes DISTINCT
 - Any window functions
 - SELECT DISTINCT or HAVING clauses
 - External tables, such as datashares and federated tables
 - Other materialized views

It isn't guaranteed that a query that meets the criteria will initiate the creation of an automated materialized view. The system determines from which candidates to create a view, based on its expected benefit to the workload and cost in resources to maintain, which includes the cost to the system to refresh. Each resulting materialized view is usable by automatic query rewriting.

- Even though AutoMV might be initiated by a subquery or individual legs of set operators, the resulting materialized view won't contain subqueries or set operators.
- To determine if AutoMV was used for queries, view the EXPLAIN plan and look for %_auto_mv_% in the output. For more information, see [EXPLAIN](#).

Automated materialized views limitations

Following are limitations for working with automated materialized views:

- *Maximum number of AutoMVs* - The limit of automated materialized views is 200 per database in the cluster.
- *Storage space and capacity* - An important characteristic of AutoMV is that it is performed using spare background cycles to help achieve that user workloads are not impacted. If the cluster is busy or running out of storage space, AutoMV ceases its activity. Specifically, at 80% of total cluster capacity, no new automated materialized views are created. At 90% of total capacity, they may be dropped to facilitate that user workloads continue without performance degradation. For more information about determining cluster capacity, see [STV_NODE_STORAGE_CAPACITY \(p. 1590\)](#).

Billing for automated materialized views

Amazon Redshift's automatic optimization capability creates and refreshes automated materialized views. There is no charge for compute resources for this process. Storage of automated materialized views is charged at the regular rate for storage. For more information, see [Amazon Redshift pricing](#).

Additional resources

The following blog post provides further explanation regarding automated materialized views. It details how they're created, maintained, and dropped. It also explains the underlying algorithms that drive these decisions: [Optimize your Amazon Redshift query performance with automated materialized views](#).

This video begins with an explanation of materialized views and shows how they improve performance and conserve resources. It then provides an in-depth explanation of automated materialized views with a process-flow animation and a live demonstration.

Using a user-defined function (UDF) in a materialized view

You can use a scalar UDF in an Amazon Redshift materialized view. Define these either in python or SQL and reference them in the materialized view definition.

Referencing a UDF in a materialized view

The following procedure shows how to use UDFs that perform simple arithmetic comparisons, in a materialized-view definition.

1. Create a table to use in the materialized-view definition.

```
CREATE TABLE base_table (a int, b int);
```

2. Create a scalar user-defined function in python that returns a boolean value indicating whether an integer is larger than a comparison integer.

```
CREATE OR REPLACE FUNCTION udf_python_bool(x1 int, x2 int) RETURNS bool IMMUTABLE
AS $$
    return x1 > x2
$$ LANGUAGE plpython;
```

Optionally, create a functionally similar UDF with SQL, which you can use to compare results with the first.

```
CREATE OR REPLACE FUNCTION udf_sql_bool(int, int) RETURNS bool IMMUTABLE
AS $$
    select $1 > $2;
$$ LANGUAGE SQL;
```

3. Create a materialized view that selects from the table you created and references the UDF.

```
CREATE MATERIALIZED VIEW mv_python_udf AS SELECT udf_python_bool(a, b) AS a FROM
base_table;
```

Optionally, you can create a materialized view that references the SQL UDF.

```
CREATE MATERIALIZED VIEW mv_sql_udf AS SELECT udf_sql_bool(a, b) AS a FROM base_table;
```

4. Add data to the table and refresh the materialized view.

```
INSERT INTO base_table VALUES (1,2), (1,3), (4,2);
```

```
REFRESH MATERIALIZED VIEW mv_python_udf;
```

Optionally, you can refresh the materialized view that references the SQL UDF.

```
REFRESH MATERIALIZED VIEW mv_sql_udf;
```

5. Query data from your materialized view.

```
SELECT * FROM mv_python_udf ORDER BY a;
```

The results of the query are the following:

```
a
-----
false
false
true
```

This returns true for the last set of values because the value for column a (4) is greater than the value for column b (2).

6. Optionally, you can query the materialized view that references the SQL UDF. The results for the SQL function match the results from the Python version.

```
SELECT * FROM mv_sql_udf ORDER BY a;
```

The results of the query are the following:

```
a
-----
false
false
true
```

This returns true for the last set of values to compare.

7. Use a DROP statement with CASCADE to drop the user-defined function and the materialized view that references it.

```
DROP FUNCTION udf_python_bool(int, int) CASCADE;
```

```
DROP FUNCTION udf_sql_bool(int, int) CASCADE;
```

Streaming ingestion

Previously, loading data from a streaming service like Amazon Kinesis into Amazon Redshift included several steps. These included connecting the stream to Amazon Kinesis Data Firehose and waiting for Kinesis Data Firehose to stage the data in Amazon S3, using various-sized batches at varying-length buffer intervals. After this, Kinesis Data Firehose initiated a COPY command to load the data from Amazon S3 to a table in Redshift.

Rather than staging in Amazon S3, streaming ingestion provides low-latency, high-speed ingestion of stream data from [Amazon Kinesis Data Streams](#) and [Amazon Managed Streaming for Apache Kafka](#) into an Amazon Redshift materialized view. Thus, it lowers the time it takes to access data and it reduces storage cost. You can configure streaming ingestion for your Amazon Redshift cluster or for Amazon Redshift Serverless and create a materialized view, using SQL statements, as described in [Creating materialized views in Amazon Redshift \(p. 198\)](#). After that, using materialized view refresh, you can ingest hundreds of megabytes of data per second. This results in fast access to external data that is quickly refreshed.

Data flow

An Amazon Redshift provisioned cluster is the stream consumer. A materialized view is the landing area for data read from the stream, which is processed as it arrives. For instance, JSON values can be consumed and mapped to the materialized view's data columns, using familiar SQL. When the materialized view is refreshed, Amazon Redshift compute nodes allocate each Kinesis data shard or Kafka partition to a compute slice. Each slice consumes data from the allocated shards until the view reaches parity with the SEQUENCE_NUMBER for the Kinesis stream or last Offset for the Kafka topic. Subsequent materialized view refreshes read data from the last SEQUENCE_NUMBER of the previous refresh until it reaches parity with the stream or topic data.

Streaming ingestion use cases

Use cases for Amazon Redshift streaming ingestion involve working with data that is generated continually (streamed) and must be processed within a short period (latency) of its generation. This is called near real-time analytics. Sources of data can vary, and include IoT devices, system telemetry data, or clickstream data from a busy website or application.

Streaming ingestion considerations

The following are important considerations and best practices for performance and billing as you set up your streaming ingestion environment.

- *Auto refresh usage and activation* - Auto refresh queries for a materialized view or views are treated as any other user workload. Auto refresh loads data from the stream as it arrives. Auto refresh can be turned on explicitly for a materialized view created for streaming ingestion. To do this, specify AUTO REFRESH in the materialized view definition. Manual refresh is the default. To specify auto refresh for an existing materialized view for streaming ingestion, you can run ALTER MATERIALIZED VIEW to turn it on. For more information, see [CREATE MATERIALIZED VIEW](#) or [ALTER MATERIALIZED VIEW](#).
- *Streaming ingestion and Amazon Redshift Serverless* - The same setup and configuration instructions that apply to Amazon Redshift streaming ingestion on a provisioned cluster also apply to streaming ingestion on Amazon Redshift Serverless. It's important to size Amazon Redshift Serverless with the necessary level of RPUs to support streaming ingestion with auto refresh and other workloads. For more information, see [Billing for Amazon Redshift Serverless](#).
- *Amazon Redshift nodes in a different availability zone than the Amazon MSK cluster* - When you configure streaming ingestion, Amazon Redshift attempts to connect to an Amazon MSK cluster in the same Availability Zone, if rack awareness is enabled for Amazon MSK. If all of your nodes are in different Availability Zones than your Amazon Redshift cluster, you can incur cross Availability Zone data-transfer cost. To avoid this, keep at least one Amazon MSK broker cluster node in the same AZ as your Amazon Redshift cluster.
- *Refresh start location* - After creating a materialized view, its initial refresh starts from the TRIM_HORIZON of a Kinesis stream, or from offset 0 of an Amazon MSK topic.
- *Data formats* - Supported data formats are limited to those that can be converted from VARBYTE. For more information, see [VARBYTE type \(p. 595\)](#) and [VARBYTE operators \(p. 1045\)](#).
- *Streaming to multiple materialized views* - In Amazon Redshift, we recommend in most cases that you land the data for each stream in a single materialized view. However, it is possible to ingest a stream and land the data in multiple materialized views. For instance, a use case where you ingest a stream containing sports data, but you organize data for each sport into a separate materialized views.

Note that when you ingest data into and refresh multiple materialized views, there can be higher egress costs, specifically for reading data from the streaming provider. Additionally, higher resource use for reading into more than one materialized view can impact other workloads. Also note bandwidth, throughput and performance limitations for your streaming provider. For more information about pricing for data streams, see [Kinesis Data Streams pricing](#) and [Amazon Managed Streaming for Apache Kafka pricing](#).

Limitations

Feature or behavior	Description
Kafka topic length limit	It isn't possible to use a Kafka topic with a name longer than 128 characters (not including quotation marks). For more information, see Names and identifiers .
Incremental refreshes and JOINs on a materialized view	The materialized view must be incrementally maintainable. Full recompute is not possible for Kinesis or Amazon MSK because they don't preserve stream or topic history past 24 hours or 7 days, by default. You can set longer data retention periods in Kinesis or Amazon MSK. However, this can result in more maintenance and cost. Additionally, JOINs are not currently supported on

Feature or behavior	Description
	materialized views created on a Kinesis stream, or on an Amazon MSK topic. After creating a materialized view on your stream or topic, you can create another materialized view in order to join your streaming materialized view to other materialized views, tables, or views. For more information, see REFRESH MATERIALIZED VIEW .
Record parsing	Amazon Redshift streaming ingestion doesn't support parsing records that have been aggregated by the Kinesis Producer Library (KPL Key Concepts - Aggregation). The aggregated records are ingested, but are stored as binary protocol buffer data. (See Protocol buffers for more information.) Depending on how you push data to Kinesis, you may need to turn off this feature.
Decompression	VARBYTE does not currently support any decompression methods. Because of this, records containing compressed data can't be queried inside Amazon Redshift. Decompress your data before pushing it into the Kinesis stream or Amazon MSK topic.
Maximum record size	The maximum size of any record field Amazon Redshift can ingest from Kinesis or Amazon MSK is slightly less than 1MB. The following points detail the behavior: <ul style="list-style-type: none"> Maximum VARBYTE length - The VARBYTE type supports data to a maximum length of 1,024,000 bytes. Because Kinesis limits payloads to 1MB, after Base64 encoding, all Kinesis data can be ingested by Amazon Redshift. Message limits - Default Amazon MSK configuration limits messages to 1MB. Additionally, if a message includes headers, the amount of data is limited to 1,048,470 bytes. With default settings, there are no problems with ingestion. However, you can change the maximum message size for Kafka, and therefore Amazon MSK, to a larger value. In this case, it might be possible for the key/value field of a Kafka record, or the header, to exceed the size limit. These records can cause an error and are not ingested.
Error records	In each case where a record can't be ingested to Amazon Redshift because the size of the data exceeds the maximum size, that record is skipped. Materialized view refresh still succeeds, in this case, and a segment of each error record is written to the SYS_STREAM_SCAN_ERRORS (p. 1490) system table. Errors that result from business logic, such as an error in a calculation or an error resulting from a type conversion, are not skipped. Test the logic carefully, before you add logic to your materialized view definition, to avoid these.

Getting started with streaming ingestion from Amazon Kinesis Data Streams

Setting up Amazon Redshift streaming ingestion involves creating an external schema that maps to the streaming data source and creating a materialized view that references the external schema. Amazon

Redshift streaming ingestion supports Kinesis Data Streams as a source. As such, you must have a Kinesis Data Streams source available before configuring streaming ingestion. If you don't have a source, follow the instructions in the Kinesis documentation at [Getting Started with Amazon Kinesis Data Streams](#) or create one on the console using the instructions at [Creating a Stream via the AWS Management Console](#).

Amazon Redshift streaming ingestion uses a materialized view, which is updated directly from the stream when REFRESH is run. The materialized view maps to the stream data source. You can perform filtering and aggregations on the stream data as part of the materialized-view definition. Your streaming ingestion materialized view (the *base* materialized view) can reference only one stream, but you can create additional materialized views that join with the base materialized view and with other materialized views or tables.

Note

Streaming ingestion and Amazon Redshift Serverless - The configuration steps in this topic apply both to provisioned Amazon Redshift clusters and to Amazon Redshift Serverless. For more information, see [Streaming ingestion considerations \(p. 209\)](#).

Assuming you have a Kinesis Data Streams stream available, the first step is to define a schema in Amazon Redshift with CREATE EXTERNAL SCHEMA and to reference a Kinesis Data Streams resource. Following that, to access data in the stream, define the STREAM in a materialized view. You can store stream records in the semi-structured SUPER format, or define a schema that results in data converted to Redshift data types. When you query the materialized view, the returned records are a point-in-time view of the stream.

1. Create an IAM role with a trust policy that allows your Amazon Redshift cluster to assume the role. For information about how to configure the trust policy for the IAM role, see [Authorizing Amazon Redshift to access other AWS services on your behalf](#). After it is created, the role should have the following IAM policy, which provides permission for communication with the Amazon Kinesis data stream.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ReadStream",  
            "Effect": "Allow",  
            "Action": [  
                "kinesis:DescribeStreamSummary",  
                "kinesis:GetShardIterator",  
                "kinesis:GetRecords",  
                "kinesis:DescribeStream"  
            ],  
            "Resource": "arn:aws:kinesis:*:0123456789:stream/*"  
        },  
        {  
            "Sid": "ListStream",  
            "Effect": "Allow",  
            "Action": [  
                "kinesis>ListStreams",  
                "kinesis>ListShards"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

2. Check your VPC and verify that your Amazon Redshift cluster has a route to get to the Kinesis Data Streams endpoints over the internet using a NAT gateway or internet gateway. If you want traffic between your Amazon Redshift cluster and Kinesis Data Streams to remain within the AWS network, consider using a Kinesis Interface VPC Endpoint. For more information, see [Using Amazon Kinesis Data Streams Kinesis Data Streams with Interface VPC Endpoints](#).

-
3. In Amazon Redshift, create an external schema to map the data from Kinesis to a schema.

```
CREATE EXTERNAL SCHEMA kds
FROM KINESIS
IAM_ROLE { default | 'iam-role-arn' };
```

Streaming ingestion for Kinesis Data Streams doesn't require an authentication type. It uses the IAM role defined in the CREATE EXTERNAL SCHEMA statement for making Kinesis Data Streams requests.

Optional: Use the REGION keyword to specify the region where the Amazon Kinesis Data Streams or Amazon MSK stream resides.

```
CREATE EXTERNAL SCHEMA kds
FROM KINESIS
REGION 'us-west-2'
IAM_ROLE { default | 'iam-role-arn' };
```

In this sample, the region specifies the location of the source stream. The IAM_ROLE is a sample.

4. Create a materialized view to consume the stream data. The following example defines a materialized view with JSON source data. Note that the following view validates the data is valid JSON source . Kinesis stream names are case sensitive and can contain both uppercase and lowercase letters. To use case-sensitive identifiers, you can set the configuration setting enable_case_sensitive_identifier to true at either the session or cluster level. For more information, see [Names and identifiers](#) and [enable_case_sensitive_identifier](#).

```
CREATE MATERIALIZED VIEW my_view AUTO REFRESH YES AS
SELECT approximate_arrival_timestamp,
JSON_PARSE(kinesis_data) as Data
FROM kds.my_stream_name
WHERE CAN_JSON_PARSE(kinesis_data);
```

To turn on auto refresh, use AUTO REFRESH YES. The default behavior is manual refresh.

Metadata columns include the following:

Metadata column	Data type	Description
approximate_arrival_timestamp	timestamp without time zone	The approximate time that the record was inserted into the Kinesis stream
partition_key	varchar(256)	The key used by Kinesis to assign the record to a shard
shard_id	char(20)	The unique identifier of the shard within the stream from which the record was retrieved
sequence_number	varchar(128)	The unique identifier of the record from the Kinesis shard
refresh_time	timestamp without time zone	The time the refresh started
kinesis_data	varbyte	The record from the Kinesis stream

5. Refresh the view, which invokes Amazon Redshift to read from the stream and load data into the materialized view.

```
REFRESH MATERIALIZED VIEW my_view;
```

6. Query data in the materialized view.

```
select * from my_view;
```

Getting started with streaming ingestion from Amazon Managed Streaming for Apache Kafka

The purpose of Amazon Redshift streaming ingestion is to simplify the process for directly ingesting stream data from a streaming service into Amazon Redshift. This works with Amazon MSK and Amazon MSK Serverless, and with Kinesis. Amazon Redshift streaming ingestion removes the need to stage a Kinesis Data Streams stream or an Amazon MSK topic in Amazon S3 before ingesting the stream data into Amazon Redshift.

On a technical level, streaming ingestion, both from Amazon Kinesis Data Streams and Amazon Managed Streaming for Apache Kafka, provides low-latency, high-speed ingestion of stream or topic data into an Amazon Redshift materialized view. Following setup, using materialized view refresh, you can take in large data volumes.

Set up Amazon Redshift streaming ingestion for Amazon MSK by performing the following steps:

1. Create an external schema that maps to the streaming data source.
2. Create a materialized view that references the external schema.

You must have an Amazon MSK source available, before configuring Amazon Redshift streaming ingestion. If you do not have a source, follow the instructions at [Getting Started Using Amazon MSK](#).

Note

Streaming ingestion and Amazon Redshift Serverless - The configuration steps in this topic apply both to provisioned Amazon Redshift clusters and to Amazon Redshift Serverless. For more information, see [Streaming ingestion considerations \(p. 209\)](#).

Setting up IAM and performing streaming ingestion from Kafka

Assuming you have an Amazon MSK cluster available, the first step is to define a schema in Amazon Redshift with CREATE EXTERNAL SCHEMA and to reference the Kafka topic as the data source. Following that, to access data in the topic, define the STREAM in a materialized view. You can store records from your topic in the semi-structured SUPER format, or define a schema that results in data converted to Amazon Redshift data types. When you query the materialized view, the returned records are a point-in-time view of the topic.

1. Create an IAM role with a trust policy that allows your Amazon Redshift cluster to assume the role. For information about how to configure the trust policy for the IAM role, see [Authorizing Amazon Redshift to access other AWS services on your behalf](#). After it's created, the role should have the following IAM policy, which provides permission for communication with the Amazon MSK cluster. The policy you need depends on the authentication method used on your cluster, if you use Amazon MSK. See [Authentication and Authorization for Apache Kafka APIs](#) for authentication methods available in Amazon MSK.

An IAM policy for Amazon MSK using unauthenticated access:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Allow",  
            "Action": [  
                "kafka:GetBootstrapBrokers"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

An IAM policy for Amazon MSK when using IAM authentication:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "MSKIAMpolicy",  
            "Effect": "Allow",  
            "Action": [  
                "kafka-cluster:ReadData",  
                "kafka-cluster:DescribeTopic",  
                "kafka-cluster:Connect"  
            ],  
            "Resource": [  
                "arn:aws:kafka:*:0123456789:cluster/*/*",  
                "arn:aws:kafka:*:0123456789:topic/*/*/*"  
            ]  
        },  
        {  
            "Sid": "MSKPolicy",  
            "Effect": "Allow",  
            "Action": [  
                "kafka:GetBootstrapBrokers"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

2. Check your VPC and verify that your Amazon Redshift cluster has a route to get to your Amazon MSK cluster. The inbound security group rules for your Amazon MSK cluster should allow your Amazon Redshift cluster's security group. The ports you specify depend on the authentication method used for your cluster, if you use Amazon MSK. For more information, see [Port information](#) and [Access from within AWS but outside the VPC](#).

Note that client authentication with TLS isn't supported for streaming ingestion. For more information, see [Limitations](#).

3. Enable enhanced VPC routing on your Amazon Redshift cluster. For more information, see [Enabling enhanced VPC routing](#).
4. In Amazon Redshift, create an external schema to map to the Amazon MSK cluster.

```
CREATE EXTERNAL SCHEMA MySchema  
FROM MSK  
IAM_ROLE { default | 'iam-role-arn' }  
AUTHENTICATION { none | iam }  
CLUSTER_ARN 'msk-cluster-arn';
```

In the FROM clause, Amazon MSK denotes that the schema maps data from Managed Kafka Services.

Streaming ingestion for Amazon MSK provides the following authentication types, when you create the external schema:

- **none** – Specifies that there is no authentication step.
- **iam** – Specifies IAM authentication. When you choose this, make sure that the IAM role has permissions for IAM authentication.

Additional Amazon MSK authentication methods, such as TLS authentication or a username and password, aren't supported for streaming ingestion.

CLUSTER_ARN specifies the Amazon MSK cluster that you're streaming from.

5. Create a materialized view to consume the data from the topic. The following example defines a materialized view with JSON source data. Note that the following view validates that the data is valid JSON and utf8. Kafka topic names are case sensitive and can contain both uppercase and lowercase letters. To use case-sensitive identifiers, you can set the configuration `enable_case_sensitive_identifier` to true at either the session or cluster level. For more information, see [Names and identifiers](#) and [r_enable_case_sensitive_identifier](#).

```
CREATE MATERIALIZED VIEW MyView AUTO REFRESH YES AS
SELECT "kafka_partition",
       "kafka_offset",
       "kafka_timestamp_type",
       "kafka_timestamp",
       "kafka_key",
       JSON_PARSE("kafka_value") as Data,
       "kafka_headers"
FROM MySchema.MyTopic;
```

To turn on auto refresh, use AUTO REFRESH YES. The default behavior is manual refresh.

Metadata columns include the following:

Metadata column	Data type	Description
kafka_partition	bigint	Partition id of the record from the Kafka topic
kafka_offset	bigint	Offset of the record in the Kafka topic for a given partition
kafka_timestamp_type	char(1)	Type of timestamp used in the Kafka record: <ul style="list-style-type: none"> • <i>C</i> – Record creation time (CREATE_TIME) on the client side • <i>L</i> – Record append time (LOG_APPEND_TIME) on the Kafka server side • <i>U</i> – Record creation time is not available (NO_TIMESTAMP_TYPE)

Metadata column	Data type	Description
kafka_timestamp	timestamp without time zone	The timestamp value for the record
kafka_key	varbyte	The key of the Kafka record
kafka_value	varbyte	The record received from Kafka
kafka_headers	super	The header of the record received from Kafka
refresh_time	timestamp without time zone	The time the refresh started

6. Refresh the view, which invokes Amazon Redshift to read from the topic and load data into the materialized view.

```
REFRESH MATERIALIZED VIEW MyView;
```

7. Query data in the materialized view.

```
select * from MyView;
```

The materialized view is updated directly from the topic when REFRESH is run. You create a materialized view that maps to the Kafka topic data source. You can perform filtering and aggregations on the data as part of the materialized view definition. Your streaming ingestion materialized view (base materialized view) can reference only one Kafka topic, but you can create additional materialized views that join with the base materialized view and with other materialized views or tables.

For more information about limitations for streaming ingestion, see [Limitations \(p. 209\)](#).

Electric vehicle station-data streaming ingestion tutorial, using Kinesis

This procedure demonstrates how to ingest data from a Kinesis stream named `ev_station_data`, which contains consumption data from different EV charging stations, in JSON format. The schema is well defined. The example shows how to store the data as raw JSON and also how to convert the JSON data to Amazon Redshift data types as it's ingested.

Producer setup

1. Using Amazon Kinesis Data Streams, follow the steps to create a stream named `ev_station_data`. Choose **On-demand** for the **Capacity mode**. For more information, see [Creating a Stream via the AWS Management Console](#).
2. The [Amazon Kinesis Data Generator](#) can help you generate test data for use with your stream. Follow the steps detailed in the tool to get started, and use the following data template for generating your data:

```
{
    "_id" : "{{random.uuid}}",
    "clusterID": "{{random.number(
        {
            "min":1,
            "max":50
        }
    )}}
```

```

        )}}",
    "connectionTime": "{{date.now("YYYY-MM-DD HH:mm:ss")}}",
    "kWhDelivered": "{{commerce.price}}",
    "stationID": "{{random.number(
        {
            "min":1,
            "max":467
        }
    )}}",
    "spaceID": "{{random.word}}-{{random.number(
        {
            "min":1,
            "max":20
        }
    )}}",
    "timezone": "America/Los_Angeles",
    "userID": "{{random.number(
        {
            "min":1000,
            "max":500000
        }
    )}}"
}

```

Each JSON object in the stream data has the following properties:

```
{
    "_id": "12084f2f-fc41-41fb-a218-8cc1ac6146eb",
    "clusterID": "49",
    "connectionTime": "2022-01-31 13:17:15",
    "kWhDelivered": "74.00",
    "stationID": "421",
    "spaceID": "technologies-2",
    "timezone": "America/Los_Angeles",
    "userID": "482329"
}
```

Amazon Redshift setup

These steps show you how to configure the materialized view to ingest data.

1. Create an external schema to map the data from Kinesis to a Redshift object.

```
CREATE EXTERNAL SCHEMA evdata FROM KINESIS
IAM_ROLE 'arn:aws:iam::0123456789:role/redshift-streaming-role';
```

For information about how to configure the IAM role, see [Getting started with streaming ingestion from Amazon Kinesis Data Streams \(p. 210\)](#).

2. Create a materialized view to consume the stream data. The following examples show both methods of defining materialized views to ingest the JSON source data.

First, store stream records in semi-structured SUPER format. In this example, the JSON source is stored in Redshift without converting to Redshift types.

```
CREATE MATERIALIZED VIEW ev_station_data AS
    SELECT approximate_arrival_timestamp,
           partition_key,
           shard_id,
           sequence_number,
           json_parse(kinesis_data) as payload
    FROM evdata."ev_station_data" WHERE can_json_parse(kinesis_data);
```

In contrast, in the following materialized view definition, the materialized view has a defined schema in Redshift. The materialized view is distributed on the UUID value from the stream and is sorted by the approximatearrivaltimestamp value.

```
CREATE MATERIALIZED VIEW ev_station_data_extract DISTKEY(6) sortkey(1) AUTO REFRESH YES
AS
    SELECT refresh_time,
    approximate_arrival_timestamp,
    partition_key,
    shard_id,
    sequence_number,
    json_extract_path_text(from_varbyte(kinesis_data,'utf-8'), '_id', true)::character(36)
    as ID,
    json_extract_path_text(from_varbyte(kinesis_data,'utf-8'), 'clusterID', true)::varchar(30)
    as clusterID,
    json_extract_path_text(from_varbyte(kinesis_data,'utf-8'), 'connectionTime', true)::varchar(20)
    as connectionTime,
    json_extract_path_text(from_varbyte(kinesis_data,'utf-8'), 'kWhDelivered', true)::DECIMAL(10,2)
    as kWhDelivered,
    json_extract_path_text(from_varbyte(kinesis_data,'utf-8'), 'stationID', true)::DECIMAL(10,2)
    as stationID,
    json_extract_path_text(from_varbyte(kinesis_data,'utf-8'), 'spaceID', true)::varchar(100)
    as spaceID,
    json_extract_path_text(from_varbyte(kinesis_data,'utf-8'), 'timezone', true)::varchar(30)
    as timezone,
    json_extract_path_text(from_varbyte(kinesis_data,'utf-8'), 'userID', true)::varchar(30)
    as userID
    FROM evdata."ev_station_data"
    WHERE LENGTH(kinesis_data) < 65355;
```

Query the stream

1. Query the refreshed materialized view to get usage statistics.

```
SELECT to_timestamp(connectionTime, 'YYYY-MM-DD HH24:MI:SS') as connectiontime
, SUM(kWhDelivered) AS Energy_Consumed
, count(distinct userID) AS #Users
from ev_station_data_extract
group by to_timestamp(connectionTime, 'YYYY-MM-DD HH24:MI:SS')
order by 1 desc;
```

2. View results.

connectiontime	energy_consumed	#users
2022-02-08 16:07:21+00	4139	10
2022-02-08 16:07:20+00	5571	10
2022-02-08 16:07:19+00	8697	20
2022-02-08 16:07:18+00	4408	10
2022-02-08 16:07:17+00	4257	10
2022-02-08 16:07:16+00	6861	10
2022-02-08 16:07:15+00	5643	10
2022-02-08 16:07:14+00	3677	10
2022-02-08 16:07:13+00	4673	10

Querying spatial data in Amazon Redshift

Spatial data describes the position and shape of a geometry in a defined space (a spatial reference system). Amazon Redshift supports spatial data with the GEOMETRY and GEOGRAPHY data types, which contain spatial data and optionally the data's spatial reference system identifier (SRID).

Spatial data contains geometric data that you can use to represent geographic features. Examples of this type of data include weather reports, map directions, tweets with geographic positions, store locations, and airline routes. Spatial data plays an important role in business analytics, reporting, and forecasting.

You can query spatial data with Amazon Redshift SQL functions. Spatial data contains geometric values for an object.

The GEOMETRY data type operations work on the Cartesian plane. Although the spatial reference system identifier (SRID) is stored inside the object, this SRID is merely an identifier of the coordinate system and plays no role in the algorithms used to process the GEOMETRY objects. Conversely, the operations on the GEOGRAPHY data type treat the coordinates inside objects as spherical coordinates on a spheroid. This spheroid is defined by the SRID, which references a geographic spatial reference system. By default, GEOGRAPHY data types are created with spatial reference (SRID) 4326, referencing the World Geodetic System (WGS) 84.

You can use the ST_Transform function to transform the coordinates from various spatial reference systems. After the transformation of the coordinates is done, you can also use a simple cast between the two, as long as the input GEOMETRY is encoded with the geographic SRID. This cast simply copies coordinates without any further transformation. For example:

```
SELECT ST_AsEWKT(ST_GeomFromEWKT('SRID=4326;POINT(10 20)')::geography);
```

```
st_asewkt
-----
SRID=4326;POINT(10 20)
```

To better understand the difference between GEOMETRY and GEOGRAPHY data types, consider calculating the distance between the Berlin airport (BER) and the San Francisco airport (SFO) using the World Geodetic System (WGS) 84. Using the GEOGRAPHY data type, the result is in meters. When using GEOMETRY data type with SRID 4326, the result is in degrees, which can't convert to meters because the distance of one degree depends on where on the globe geometries are located.

Calculations on the GEOGRAPHY data type are mostly used for realistic round earth calculations such as the precise area of a country without distortion. But they are far more expensive to compute. Therefore, ST_Transform can transform your coordinates to an appropriate local projected coordinate system and do the calculation on the GEOMETRY data type faster.

Using spatial data, you can run queries to do the following:

- Find the distance between two points.
- Check whether one area (polygon) contains another.
- Check whether one linestring intersects another linestring or polygon.

You can use the GEOMETRY data type to hold the values of spatial data. A GEOMETRY value in Amazon Redshift can define two-dimensional (2D), three-dimensional (3DZ), two-dimensional with a measure (3DM), and four-dimensional (4D) geometry primitive data types:

- A two-dimensional (2D) geometry is specified by two Cartesian coordinates (x, y) in a plane.
- A three-dimensional (3DZ) geometry is specified by three Cartesian coordinates (x, y, z) in space.
- A two-dimensional with measure (3DM) geometry is specified by three coordinates (x, y, m), where the first two are Cartesian coordinates in a plane and the third is a measurement.
- A four-dimensional (4D) geometry is specified by four coordinates (x, y, z, m), where the first three are Cartesian coordinates in a space and the fourth is a measurement.

For more information about geometry primitive data types, see [Well-known text representation of geometry](#) in Wikipedia.

You can use the GEOGRAPHY data type to hold the values of spatial data. A GEOGRAPHY value in Amazon Redshift can define two-dimensional (2D), three-dimensional (3DZ), two-dimensional with a measure (3DM), and four-dimensional (4D) geometry primitive data types:

- A two-dimensional (2D) geometry is specified by longitude and latitude coordinates on a spheroid.
- A three-dimensional (3DZ) geometry is specified by longitude, latitude, and altitude coordinates on a spheroid.
- A two-dimensional with measure (3DM) geometry is specified by three coordinates (longitude, latitude, measure), where the first two are angular coordinates on a sphere and the third is a measurement.
- A four-dimensional (4D) geometry is specified by four coordinates (longitude, latitude, altitude, measure), where the first three are longitude, latitude and altitude, and the fourth is a measurement.

For more information about geographic coordinate systems, see [Geographic coordinate system](#) and [Spherical coordinate system](#) in Wikipedia.

The GEOMETRY and GEOGRAPHY data types have the following subtypes:

- POINT
- LINESTRING
- POLYGON
- MULTIPOLYPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

There are Amazon Redshift SQL functions that support the following representations of geometric data:

- GeoJSON
- Well-known text (WKT)
- Extended well-known text (EWKT)
- Well-known binary (WKB) representation
- Extended well-known binary (EWKB)

You can cast between GEOMETRY and GEOGRAPHY data types.

The following SQL casts a linestring from a GEOMETRY to a GEOGRAPHY.

```
SELECT ST_AsEWKT(ST_GeomFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)')::geography);
```

```
st_asewkt
-----
SRID=4326;LINESTRING(110 40,2 3,-10 80,-7 9)
```

The following SQL casts a linestring from a GEOGRAPHY to a GEOMETRY.

```
SELECT ST_AsEWKT(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)')::geometry);
```

```
st_asewkt
-----
SRID=4326;LINESTRING(110 40,2 3,-10 80,-7 9)
```

Amazon Redshift provides many SQL functions to query spatial data. Except for the `ST_IsValid` function, spatial functions that accept a GEOMETRY object as an argument expect this GEOMETRY object to be a valid geometry. If the GEOMETRY or GEOGRAPHY object isn't valid, then the behavior of the spatial function is undefined. For more information about validity, see [Geometric validity \(p. 229\)](#).

For details about SQL functions to query spatial data, see [Spatial functions \(p. 1180\)](#).

For details about loading spatial data, see [Loading a column of the GEOMETRY or GEOGRAPHY data type \(p. 718\)](#).

Topics

- [Tutorial: Using spatial SQL functions with Amazon Redshift \(p. 222\)](#)
- [Loading a shapefile into Amazon Redshift \(p. 228\)](#)
- [Terminology for Amazon Redshift spatial data \(p. 229\)](#)
- [Limitations when using spatial data with Amazon Redshift \(p. 232\)](#)

Tutorial: Using spatial SQL functions with Amazon Redshift

This tutorial demonstrates how to use some of the spatial SQL functions with Amazon Redshift.

To do this, you query two tables using spatial SQL functions. The tutorial uses data from public datasets that correlate location data of rental accommodations with postal codes in Berlin, Germany.

Topics

- [Prerequisites \(p. 222\)](#)
- [Step 1: Create tables and load test data \(p. 223\)](#)
- [Step 2: Query spatial data \(p. 225\)](#)
- [Step 3: Clean up your resources \(p. 227\)](#)

Prerequisites

For this tutorial, you need the following resources:

- An existing Amazon Redshift cluster and database that you can access and update. In the existing cluster, you create tables, load sample data, and run SQL queries to demonstrate spatial functions. Your cluster should have at least two nodes. To learn how to create a cluster, follow the steps in [Amazon Redshift Getting Started Guide](#).
- To use the Amazon Redshift query editor, make sure that your cluster is in an AWS Region that supports the query editor. For more information, see [Querying a database using the query editor](#) in the [Amazon Redshift Management Guide](#).
- AWS credentials for your Amazon Redshift cluster that allow it to load test data from Amazon S3. For information about how to access other AWS services like Amazon S3, see [Authorizing Amazon Redshift to access AWS services](#).
- The AWS Identity and Access Management (IAM) role named mySpatialDemoRole, which has the managed policy AmazonS3ReadOnlyAccess attached to read Amazon S3 data. To create a role with permission to load data from an Amazon S3 bucket, see [Authorizing COPY, UNLOAD, and CREATE EXTERNAL SCHEMA operations using IAM roles](#) in the [Amazon Redshift Management Guide](#).
- After you create the IAM role mySpatialDemoRole, that role needs an association with your Amazon Redshift cluster. For more information on how to create that association, see [Authorizing COPY, UNLOAD, and CREATE EXTERNAL SCHEMA operations using IAM roles](#) in the [Amazon Redshift Management Guide](#).

Step 1: Create tables and load test data

The source data used by this tutorial is in files named `accommodations.csv` and `zipcodes.csv`.

The `accommodations.csv` file is open-source data from [insideairbnb.com](#). The `zipcodes.csv` file provides postal codes that are open-source data from the national statistics institute of Berlin-Brandenburg in Germany ([Amt für Statistik Berlin-Brandenburg](#)). Both data sources are provided under a Creative Commons license. The data is limited to the Berlin, Germany, region. These files are located in an Amazon S3 public bucket to use with this tutorial.

You can optionally download the source data from the following Amazon S3 links:

- [Source data for the accommodations table](#).
- [Source data for the zipcode table](#).

Use the following procedure to create tables and load test data.

To create tables and load test data

1. Open the Amazon Redshift query editor. For more information on working with the query editor, see [Querying a database using the query editor](#) in the [Amazon Redshift Management Guide](#).
2. Drop any tables used by this tutorial if they already exist in your database. For more information, see [Step 3: Clean up your resources \(p. 227\)](#).
3. Create the `accommodations` table to store each accommodation's geographical location (longitude and latitude), the name of the listing, and other business data.

This tutorial explores room rentals in Berlin, Germany. The `shape` column stores geographic points of the location of accommodations. The other columns contain information about the rental.

To create the `accommodations` table, run the following SQL statement in the Amazon Redshift query editor.

```
CREATE TABLE public.accommodations (
    id INTEGER PRIMARY KEY,
    shape GEOMETRY,
```

```

    name VARCHAR(100),
    host_name VARCHAR(100),
    neighbourhood_group VARCHAR(100),
    neighbourhood VARCHAR(100),
    room_type VARCHAR(100),
    price SMALLINT,
    minimum_nights SMALLINT,
    number_of_reviews SMALLINT,
    last_review DATE,
    reviews_per_month NUMERIC(8,2),
    calculated_host_listings_count SMALLINT,
    availability_365 SMALLINT
);

```

4. Create the zipcode table in the query editor to store Berlin postal codes.

A *postal code* is defined as a polygon in the wkb_geometry column. The rest of the columns describe additional spatial metadata about the postal code.

To create the zipcode table, run the following SQL statement in the Amazon Redshift query editor.

```

CREATE TABLE public.zipcode (
    ogc_field INTEGER PRIMARY KEY NOT NULL,
    wkb_geometry GEOMETRY,
    gml_id VARCHAR(256),
    spatial_name VARCHAR(256),
    spatial_alias VARCHAR(256),
    spatial_type VARCHAR(256)
);

```

5. Load the tables using sample data.

The sample data for this tutorial is provided in an Amazon S3 bucket that allows read access to all authenticated AWS users. Make sure that you provide valid AWS credentials that permit access to Amazon S3.

To load test data to your tables, run the following COPY commands. Replace *account-number* with your own AWS account number. The segment of the credentials string that is enclosed in single quotation marks can't contain any spaces or line breaks.

```

COPY public.accommodations
FROM 's3://redshift-downloads/spatial-data/accommodations.csv'
DELIMITER ';'
IGNOREHEADER 1 REGION 'us-east-1'
CREDENTIALS 'aws_iam_role=arn:aws:iam::account-number:role/mySpatialDemoRole';

```

```

COPY public.zipcode
FROM 's3://redshift-downloads/spatial-data/zipcode.csv'
DELIMITER ';'
IGNOREHEADER 1 REGION 'us-east-1'
CREDENTIALS 'aws_iam_role=arn:aws:iam::account-number:role/mySpatialDemoRole';

```

6. Verify that each table loaded correctly by running the following commands.

```
select count(*) from accommodations;
```

```
select count(*) from zipcode;
```

The following results show the number of rows in each table of test data.

Table name	Rows
accommodations	22,248
zipcode	190

Step 2: Query spatial data

After your tables are created and loaded, you can query them using SQL SELECT statements. The following queries demonstrate some of the information that you can retrieve. You can write many other queries that use spatial functions to satisfy your needs.

To query spatial data

1. Query to get the count of the total number of listings stored in the accommodations table, as shown following. The spatial reference system is World Geodetic System (WGS) 84, which has the unique spatial reference identifier 4326.

```
SELECT count(*) FROM public.accommodations WHERE ST_SRID(shape) = 4326;
```

```
count
-----
22248
```

2. Fetch the geometry objects in well-known text (WKT) format with some additional attributes. Additionally, you can validate if this postal code data is also stored in World Geodetic System (WGS) 84, which uses the spatial reference ID (SRID) 4326. Spatial data must be stored in the same spatial reference system to be interoperable.

```
SELECT ogc_field, spatial_name, spatial_type, ST_SRID(wkb_geometry),
       ST_AsText(wkb_geometry)
  FROM public.zipcode
 ORDER BY spatial_name;
```

```
ogc_field  spatial_name  spatial_type  st_srid  st_astext
-----
0          10115        Polygon       4326     POLYGON((...))
4          10117        Polygon       4326     POLYGON((...))
8          10119        Polygon       4326     POLYGON((...))
...
(190 rows returned)
```

3. Select the polygon of Berlin Mitte (10117), a borough of Berlin, in GeoJSON format, its dimension, and the number of points in this polygon.

```
SELECT ogc_field, spatial_name, ST_AsGeoJSON(wkb_geometry), ST_Dimension(wkb_geometry),
       ST_NPoints(wkb_geometry)
  FROM public.zipcode
 WHERE spatial_name='10117';
```

ogc_field	spatial_name	spatial_type	st_dimension
st_npoint	4	10117 {"type": "Polygon", "coordinates": [[[...]]]}	2

4. Run the following SQL command to view how many accommodations are within 500 meters of the Brandenburg Gate.

```
SELECT count(*)
FROM public.accommodations
WHERE ST_DistanceSphere(shape, ST_GeomFromText('POINT(13.377704 52.516431)', 4326)) < 500;
```

count

29

5. Get the rough location of the Brandenburg Gate from data stored in the accommodations that are listed as nearby by running the following query.

This query requires a subselect. It leads to a different count because the requested location is not the same as the previous query because it is closer to the accommodations.

```
WITH poi(loc) as (
    SELECT st_astext(shape) FROM accommodations WHERE name LIKE '%brandenburg gate%'
)
SELECT count(*)
FROM accommodations a, poi p
WHERE ST_DistanceSphere(a.shape, ST_GeomFromText(p.loc, 4326)) < 500;
```

count

60

6. Run the following query to show the details of all accommodations around the Brandenburg Gate, ordered by price in descending order.

```
SELECT name, price, ST_AsText(shape)
FROM public.accommodations
WHERE ST_DistanceSphere(shape, ST_GeomFromText('POINT(13.377704 52.516431)', 4326)) < 500
ORDER BY price DESC;
```

name	price	st_astext
DUPLEX APARTMENT/PENTHOUSE in 5* LOCATION! 7583 52.5159819722552)	300	POINT(13.3826510209548
DUPLEX-PENTHOUSE IN FIRST LOCATION! 7582 52.5135918444834)	300	POINT(13.3799997083855
...		
(29 rows returned)		

7. Run the following query to retrieve the most expensive accommodation with its postal code.

SELECT

```
a.price, a.name, ST_AsText(a.shape),
z.spatial_name, ST_AsText(z.wkb_geometry)
FROM accommodations a, zipcode z
WHERE price = 9000 AND ST_Within(a.shape, z.wkb_geometry);
```

price	name	st_astext
spatial_name	st_astext	
9000	Ueber den Dächern Berlins Zentrum	POINT(13.334436985013 52.4979779501538)
10777		POLYGON((13.3318284987227 52.4956021172799, ...))

8. Calculate the maximum, minimum, or median price of accommodations by using a subquery.

The following query lists the median price of accommodations by postal code.

```
SELECT
    a.price, a.name, ST_AsText(a.shape),
    z.spatial_name, ST_AsText(z.wkb_geometry)
FROM accommodations a, zipcode z
WHERE
    ST_Within(a.shape, z.wkb_geometry) AND
    price = (SELECT median(price) FROM accommodations)
ORDER BY a.price;
```

price	name	st_astext	
spatial_name	st_astext		
45	"Cozy room Berlin-Mitte"	POINT(13.3864349535358 52.5292016386514)	10115
		POLYGON((13.3658598465795 52.535659581048, ...))	
...			
(723 rows returned)			

9. Run the following query to retrieve the number of accommodations listed in Berlin. To find the hot spots, these are grouped by postal code and sorted by the amount of supply.

```
SELECT z.spatial_name as zip, count(*) as numAccommodations
FROM public.accommodations a, public.zipcode z
WHERE ST_Within(a.shape, z.wkb_geometry)
GROUP BY zip
ORDER BY numAccommodations DESC;
```

zip	numaccommodations
10245	872
10247	832
10437	733
10115	664
...	
(187 rows returned)	

Step 3: Clean up your resources

Your cluster continues to accrue charges as long as it's running. When you have completed this tutorial, you can delete your sample cluster.

If you want to keep the cluster but recover the storage used by the test data tables, run the following commands to delete the tables.

```
drop table public.accommodations cascade;
```

```
drop table public.zipcode cascade;
```

Loading a shapefile into Amazon Redshift

You can use the COPY command to ingest Esri shapefiles stored in Amazon S3 into Amazon Redshift tables. A *shapefile* stores the geometric location and attribute information of geographic features in a vector format. The shapefile format can spatially describe spatial objects such as points, lines, and polygons. For more information about a shapefile, see [Shapefile](#) in Wikipedia.

The COPY command supports the data format parameter SHAPEFILE. By default, the first column of the shapefile is either a GEOMETRY or IDENTITY column. All subsequent columns follow the order specified in the shapefile. However, the target table doesn't need to be in this exact layout because you can use COPY column mapping to define the order. For information about the COPY command shapefile support, see [SHAPEFILE \(p. 698\)](#).

In some cases, the resulting geometry size might be greater than the maximum for storing a geometry in Amazon Redshift. If so, you can use the COPY option SIMPLIFY or SIMPLIFY AUTO to simplify the geometries during ingestion as follows:

- Specify SIMPLIFY *tolerance* to simplify all geometries during ingestion using the Ramer-Douglas-Peucker algorithm and the given tolerance.
- Specify SIMPLIFY AUTO without tolerance to simplify only geometries that are larger than the maximum size using the Ramer-Douglas-Peucker algorithm. This approach calculates the minimum tolerance that is large enough to store the object within the maximum size limit.
- Specify SIMPLIFY AUTO *max_tolerance* to simplify only geometries that are larger than the maximum size using the Ramer-Douglas-Peucker algorithm and the automatically calculated tolerance. This approach makes sure that the tolerance doesn't exceed the maximum tolerance.

For information about the maximum size of a GEOMETRY data value, see [Limitations when using spatial data with Amazon Redshift \(p. 232\)](#).

In some cases, the tolerance is low enough that the record can't shrink below the maximum size of a GEOMETRY data value. In these cases, you can use the MAXERROR option of the COPY command to ignore all or up to a certain number of ingestion errors.

The COPY command also supports loading GZIP shapefiles. To do this, specify the COPY GZIP parameter. With this option, all shapefile components must be independently compressed and share the same compression suffix.

If a projection description file (.prj) exists with the shapefile, Redshift uses it to determine the spatial reference system id (SRID). If the SRID is valid, the resulting geometry has this SRID assigned. If the SRID value associated with the input geometry does not exist, the resulting geometry has the SRID value zero. You can disable automatic detection of the spatial reference system id at the session level by using SET `read_srid_on_shapefile_ingestion` to OFF.

Query the `SVL_SPATIAL_SIMPLIFY` system view to view which records have been simplified, along with the calculated tolerance. When you specify SIMPLIFY *tolerance*, this view contains a record for each COPY operation. Otherwise, it contains a record for each simplified geometry. For more information, see [SVL_SPATIAL_SIMPLIFY \(p. 1658\)](#).

For examples of loading a shapefile, see [Loading a shapefile into Amazon Redshift \(p. 742\)](#).

Terminology for Amazon Redshift spatial data

The following terms are used to describe some Amazon Redshift spatial functions.

Bounding box

A bounding box of a geometry or geography is defined as the cross product (across dimensions) of the extents of the coordinates of all points in the geometry or geography. For two-dimensional geometries, the bounding box is a rectangle that completely includes all points in the geometry. For example, a bounding box of the polygon `POLYGON((0 0, 1 0, 0 2, 0 0))` is the rectangle that is defined by the points (0, 0) and (1, 2) as its bottom-left and top-right corners. Amazon Redshift precomputes and stores a bounding box inside a geometry to speed up geometric predicates and spatial joins. For example if the bounding boxes of two geometries don't intersect, then these two geometries can't intersect, and they can't be in the result set of a spatial join using the `ST_Intersects` predicate.

You can use spatial functions to add ([AddBBox \(p. 1183\)](#)), drop ([DropBBox \(p. 1183\)](#)), and determine support ([SupportsBBox \(p. 1276\)](#)) for a bounding box. Amazon Redshift supports the precomputation of bounding boxes for all geometry subtypes.

The following example shows how to update existing geometries in a table to store them with a bounding box. If your cluster is at cluster version 1.0.26809 or later, then all new geometries are created with a precomputed bounding box by default.

```
UPDATE my_table SET geom = AddBBox(geom) WHERE SupportsBBox(geom) = false;
```

After you update existing geometries, we recommend you run the `VACUUM` command on the updated table. For more information, see [VACUUM \(p. 1015\)](#).

To set whether geometries are encoded with a bounding box during a session, see [default_geometry_encoding \(p. 1687\)](#).

Geometric validity

Geometric algorithms used by Amazon Redshift assume that the input geometry is a valid geometry. If an input to an algorithm is not valid, then the result is undefined. The following section describes the geometric validity definitions used by Amazon Redshift for each geometry subtype.

Point

A point is considered to be valid if one of the following conditions is true:

- The point is the empty point.
- All point coordinates are finite floating point numbers.

A point can be the empty point.

Linestring

A linestring is considered to be valid if any of the following conditions are true:

- The linestring is empty; that is, it contains no points.
- All points in a nonempty linestring have coordinates that are finite floating point numbers.
- The linestring, if not empty, must be one-dimensional; that is, it can't degenerate to a point.

A linestring can't contain empty points.

A linestring can have duplicate consecutive points.

A linestring can have self-intersections.

Polygon

A polygon is considered to be valid if any of the following conditions are true:

- The polygon is empty; that is, it contains no rings.
- If not empty, a polygon is valid if all of the following conditions are true:
 - All rings of the polygon are valid. A ring is considered to be valid if all the following conditions are true:
 - All points of the ring have coordinates that are finite floating point numbers.
 - The ring is closed; that is, its first point and its last point coincide.
 - The ring doesn't have any self-intersections.
 - The ring is two-dimensional.
 - The rings of the polygon have consistent orientations. That is, if you traverse any ring, the interior of the polygon is either to your right or to your left. This means that if a polygon's exterior ring is oriented clockwise or counterclockwise, all the polygon's interior rings must have the same counterclockwise or clockwise orientation.
 - All interior rings must be within the exterior ring of the polygon.
 - Interior rings can't be nested; that is, an interior ring can't be within another interior ring.
 - Interior and exterior rings can only intersect at a finite number of points.
 - The interior of the polygon must be simply connected.

A polygon can't contain empty points.

Multipoint

A multipoint is considered to be valid if any of the following conditions are true:

- The multipoint is empty; that is, it contains no points.
- A multipoint is not empty, and all points are valid according to the point validity definition.

A multipoint can contain one or more empty points.

A multipoint can have duplicate points.

Multiinestring

A multilinestring is considered to be valid if any of the following conditions are true:

- The multilinestring is empty; that is, it contains no linestrings.
- All linestrings in a nonempty multilinestring are valid according to the linestring validity definition.

A nonempty multilinestring that consists of only empty linestrings is considered to be valid.

An empty linestring in a multilinestring doesn't affect its validity.

A multilinestring can have linestrings with duplicate consecutive points.

A multilinestring can have self-intersections.

A multilinestring can't contain empty points.

Multipolygon

A multipolygon is considered to be valid if any of the following conditions are true:

- The multipolygon doesn't contain any polygons (it is empty).
- The multipolygon is not empty and all of the following are true:
 - All polygons in the multipolygon are valid.
 - No two polygons in the multipolygon can intersect at an infinite number of points. In particular, this implies that the interior of any two polygons can't intersect and that they can only touch at a finite number of points.

An empty polygon in a multipolygon doesn't invalidate a multipolygon.

A multipolygon can't contain empty points.

Geometry collection

A geometry collection is considered to be valid if any of the following conditions are true:

- The geometry collection is empty; that is, it doesn't contain any geometries.
- All geometries in a nonempty geometry collection are valid.

This definition still applies, although in a recursive manner, for nested geometry collections.

A geometry collection can contain empty points and multipoints with empty points.

Geometric simplicity

Geometric algorithms used by Amazon Redshift assume that the input geometry is a valid geometry. If an input to an algorithm is not valid, then the simplicity check is undefined. The following section describes the geometric simplicity definitions used by Amazon Redshift for each geometry subtype.

Point

A valid point is considered to be simple if any of the following conditions are true:

- A valid point is always considered to be simple.
- An empty point is considered to be simple.

Linestring

A valid linestring is considered to be simple if any of the following conditions are true:

- The linestring is empty.
- The linestring is not empty and all of the following conditions are true:
 - It has no duplicate consecutive points.
 - It has no self-intersections, except possibly for its first point and last point, which can coincide. In other words, the linestring can't have self-intersections except at boundary points.

Polygon

A valid polygon is considered to be simple if it doesn't contain any duplicate consecutive points.

Multipoint

A valid multipoint is considered to be simple if any of the following conditions are true:

- The multipoint is empty; that is, it contains no points.
- No two nonempty points of the multipoint coincide.

Multilinestring

A valid multilinestring is considered to be simple if any of the following conditions are true:

- The multilinestring is empty.
- The multilinestring is nonempty and all of the following conditions are true:

- All its linestrings are simple.
- Any two linestrings of the multilinestring don't intersect, except at points that are boundary points of the two linestrings.

A nonempty multilinestring that consists of empty linestrings only is considered to be empty.

An empty linestring in a multilinestring doesn't affect its simplicity.

A closed linestring in a multilinestring can't intersect with any other linestring in the multilinestring.

A multilinestring can't have linestrings with duplicate consecutive points.

Multipolygon

A valid multipolygon is considered to be simple if it doesn't contain any duplicate consecutive points.

Geometry collection

A valid geometry collection is considered to be simple if any of the following conditions are true:

- The geometry collection is empty; that is, it doesn't contain any geometries.
- All geometries in a nonempty geometry collection are simple.

This definition still applies, although in a recursive manner, for nested geometry collections.

Limitations when using spatial data with Amazon Redshift

The following are limitations when using spatial data with Amazon Redshift:

- The maximum size of a GEOMETRY or GEOGRAPHY object is 1,048,447 bytes.
- Amazon Redshift Spectrum doesn't natively support spatial data. Therefore, you can't create or alter an external table with a GEOMETRY or GEOGRAPHY column.
- Data types for Python user-defined functions (UDFs) don't support the GEOMETRY or GEOGRAPHY data type.
- You can't use a GEOMETRY or GEOGRAPHY column as a sort key or a distribution key for an Amazon Redshift table.
- You can't use GEOMETRY or GEOGRAPHY columns in SQL ORDER BY, GROUP BY, or DISTINCT clauses.
- You can't use GEOMETRY or GEOGRAPHY columns in many SQL functions.
- You can't perform an UNLOAD operation on GEOMETRY or GEOGRAPHY columns into every format. You can UNLOAD GEOMETRY or GEOGRAPHY columns to text or comma-separated value (CSV) files. Doing this writes GEOMETRY or GEOGRAPHY data in hexadecimal EWKB format. If the size of the EWKB data is more than 4 MB, then a warning occurs because the data can't later be loaded into a table.
- The supported compression encoding of GEOMETRY or GEOGRAPHY data is RAW.
- When using JDBC or ODBC drivers, use customized type mappings. In this case, the client application must have information on which parameters of a ResultSet object are GEOMETRY or GEOGRAPHY objects. The ResultSetMetadata operation returns type VARCHAR.
- To copy geographic data from a SHAPFILE, first ingest into a GEOMETRY column, and then cast the objects to GEOGRAPHY objects..

The following nonspatial functions can accept an input of type GEOMETRY or GEOGRAPHY, or columns of type GEOMETRY or GEOGRAPHY:

- The aggregate function COUNT

- The conditional expressions COALESCE and NVL
- CASE expressions
- The default encoding for GEOMETRY and GEOGRAPHY is RAW. For more information, see [Compression encodings \(p. 39\)](#).

Querying data with federated queries in Amazon Redshift

By using *federated queries* in Amazon Redshift, you can query and analyze data across operational databases, data warehouses, and data lakes. With the Federated Query feature, you can integrate queries from Amazon Redshift on live data in external databases with queries across your Amazon Redshift and Amazon S3 environments. Federated queries can work with external databases in Amazon RDS for PostgreSQL, Amazon Aurora PostgreSQL-Compatible Edition, Amazon RDS for MySQL, and Amazon Aurora MySQL-Compatible Edition.

You can use federated queries to incorporate live data as part of your business intelligence (BI) and reporting applications. For example, to make data ingestion to Amazon Redshift easier you can use federated queries to do the following:

- Query operational databases directly.
- Apply transformations quickly.
- Load data into the target tables without the need for complex extract, transform, load (ETL) pipelines.

To reduce data movement over the network and improve performance, Amazon Redshift distributes part of the computation for federated queries directly into the remote operational databases. Amazon Redshift also uses its parallel processing capacity to support running these queries, as needed.

When running federated queries, Amazon Redshift first makes a client connection to the RDS or Aurora DB instance from the leader node to retrieve table metadata. From a compute node, Amazon Redshift issues subqueries with a predicate pushed down and retrieves the result rows. Amazon Redshift then distributes the result rows among the compute nodes for further processing.

Details about queries sent to the Amazon Aurora PostgreSQL database or Amazon RDS for PostgreSQL database are logged in the system view [SVL_FEDERATED_QUERY \(p. 1635\)](#).

Topics

- [Getting started with using federated queries to PostgreSQL \(p. 234\)](#)
- [Getting started using federated queries to PostgreSQL with AWS CloudFormation \(p. 235\)](#)
- [Getting started with using federated queries to MySQL \(p. 237\)](#)
- [Creating a secret and an IAM role to use federated queries \(p. 238\)](#)
- [Examples of using a federated query \(p. 240\)](#)
- [Data type differences between Amazon Redshift and supported PostgreSQL and MySQL databases \(p. 243\)](#)
- [Considerations when accessing federated data with Amazon Redshift \(p. 246\)](#)

Getting started with using federated queries to PostgreSQL

To create a federated query, you follow this general approach:

1. Set up connectivity from your Amazon Redshift cluster to your Amazon RDS or Aurora PostgreSQL DB instance.

To do this, make sure that your RDS PostgreSQL or Aurora PostgreSQL DB instance can accept connections from your Amazon Redshift cluster. We recommend that your Amazon Redshift cluster and Amazon RDS or Aurora PostgreSQL instance be in the same virtual private cloud (VPC) and subnet group. This way, you can add the security group for the Amazon Redshift cluster to the inbound rules of the security group for your RDS or Aurora PostgreSQL DB instance.

You can also set up VPC peering or other networking that allows Amazon Redshift to make connections to your RDS or Aurora PostgreSQL instance. For more information about VPC networking, see the following.

- [What is VPC peering?](#) in the *Amazon VPC Peering Guide*
- [Working with a DB instance in a VPC](#) in the *Amazon RDS User Guide*

Note

There are cases where you must enable enhanced VPC routing: For example, if your Amazon Redshift cluster is in a different VPC than your RDS or Aurora PostgreSQL instance, or if they're in the same VPC and your routes require it. Otherwise, you might receive timeout errors when you run a federated query.

2. Set up secrets in AWS Secrets Manager for your RDS PostgreSQL and Aurora PostgreSQL databases. Then reference the secrets in AWS Identity and Access Management (IAM) access policies and roles. For more information, see [Creating a secret and an IAM role to use federated queries \(p. 238\)](#).

Note

If your cluster uses enhanced VPC routing, you might need to configure an interface VPC endpoint for AWS Secrets Manager. This is necessary when the VPC and subnet of your Amazon Redshift cluster don't have access to the public AWS Secrets Manager endpoint. When you use a VPC interface endpoint, communication between the Amazon Redshift cluster in your VPC and AWS Secrets Manager is routed privately from your VPC to the endpoint interface. For more information, see [Creating an interface endpoint](#) in the *Amazon VPC User Guide*.

3. Apply the IAM role that you previously created to the Amazon Redshift cluster. For more information, see [Creating a secret and an IAM role to use federated queries \(p. 238\)](#).
4. Connect to your RDS PostgreSQL and Aurora PostgreSQL databases with an external schema. For more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#). For examples on how to use federated query, see [Examples of using a federated query \(p. 240\)](#).
5. Run your SQL queries referencing the external schema that references your RDS PostgreSQL and Aurora PostgreSQL databases.

Getting started using federated queries to PostgreSQL with AWS CloudFormation

You can use federated queries to query across operational databases. In this getting-started guide, you can automate setup by using a sample AWS CloudFormation stack to enable a federated query from an Amazon Redshift cluster to an Aurora PostgreSQL serverless database. You can get up and running quickly without having to run SQL statements to provision your resources.

The stack creates an external schema, referencing your Aurora PostgreSQL instance, which includes tables with sample data. You can query tables in the external schema from your Redshift cluster.

If instead you want to get started with federated queries by running SQL statements to set up an external schema, without using CloudFormation, see [Getting started with using federated queries to PostgreSQL \(p. 234\)](#).

Before running the CloudFormation stack for federated queries, make sure that you have an Amazon Aurora PostgreSQL-Compatible Edition serverless database with the Data API turned on. You can turn on the Data API in the database properties. If you can't find the setting, double-check that you are running a serverless instance of Aurora PostgreSQL. Also make sure that you have a Amazon Redshift cluster that uses RA3 nodes. We recommend that both the Redshift cluster and serverless Aurora PostgreSQL instance are in the same virtual private cloud (VPC) and subnet group. This way, you can add the security group for the Amazon Redshift cluster to the inbound rules of the security group for your Aurora PostgreSQL database instance.

For more information about getting started setting up an Amazon Redshift cluster, see [Getting started with Amazon Redshift](#). For more information about setting up resources with CloudFormation, see [What is AWS CloudFormation?](#). For more information about setting up an Aurora database, see [Creating an Aurora Serverless v1 DB cluster](#).

Launching a CloudFormation stack for Redshift federated queries

Use the following procedure to launch your CloudFormation stack for Amazon Redshift to enable federated queries. Before doing so, make sure you have your Amazon Redshift cluster and your serverless Aurora PostgreSQL instance set up.

To launch your CloudFormation stack for federated queries

1. Click [Launch CFN stack](#) here to launch the CloudFormation service in the AWS Management Console.

If you are prompted, sign in.

The stack creation process starts, referencing a CloudFormation template file, which is stored in Amazon S3. A CloudFormation *template* is a text file in JSON format that declares AWS resources that make up a stack.

2. Choose **Next** to enter the stack details.
3. Under **Parameters**, for the cluster, enter the following:
 - The Amazon Redshift cluster name, for example **ra3-consumer-cluster**
 - A specific database name, for example **dev**
 - The name of a database user, for example **consumeruser**

Also enter the parameters for the Aurora database, including the user, database name, port, and endpoint. We recommend using a test cluster and test serverless database, because the stack creates several database objects.

Choose **Next**.

The stack options appear.

4. Choose **Next** to accept the default settings.
5. Under **Capabilities**, choose **I acknowledge that AWS CloudFormation might create IAM resources**.
6. Choose **Create stack**.

Choose **Create stack**. CloudFormation provisions the template resources, which takes about 10 minutes, and creates an external schema.

If an error occurs while the stack is created, do the following:

- View the CloudFormation **Events** tab for information that can help you resolve the error.
- Make sure that you entered the correct name, database name, and database user name for the Redshift cluster. Also check the parameters for the Aurora PostgreSQL instance.
- Make sure that your cluster has RA3 nodes.
- Make sure that your database and Redshift cluster are in the same subnet and security group.

Querying data from the external schema

To use the following procedure, make sure that you have the required permissions for running queries on the cluster and the database described.

To query an external database with federated query

1. Connect to the Redshift database that you entered when you created the stack, using a client tool such as the Redshift query editor.
2. Query for the external schema created by the stack.

```
select * from svv_external_schemas;
```

The [SVV_EXTERNAL_SCHEMAS \(p. 1430\)](#) view returns information about available external schemas. In this case, the external schema created by the stack is returned, `myfederated_schema`. You might also have other external schemas returned, if you have any set up. The view also returns the schema's associated database. The database is the Aurora database that you entered when you created the stack. The stack adds a table to the Aurora database, called `category`, and another table called `sales`.

3. Run SQL queries on tables in the external schema that references your Aurora PostgreSQL database. The following example shows a query.

```
SELECT count(*) FROM myfederated_schema.category;
```

The `category` table returns several records. You can also return records from the `sales` table.

```
SELECT count(*) FROM myfederated_schema.sales;
```

For more examples, see [Examples of using a federated query \(p. 240\)](#).

Getting started with using federated queries to MySQL

To create a federated query to MySQL databases, you follow this general approach:

1. Set up connectivity from your Amazon Redshift cluster to your Amazon RDS or Aurora MySQL DB instance.

To do this, make sure that your RDS MySQL or Aurora MySQL DB instance can accept connections from your Amazon Redshift cluster. We recommend that your Amazon Redshift cluster and Amazon RDS or Aurora MySQL instance be in the same virtual private cloud (VPC) and subnet group. This way, you can add the security group for the Amazon Redshift cluster to the inbound rules of the security group for your RDS or Aurora MySQL DB instance.

You can also set up VPC peering or other networking that allows Amazon Redshift to make connections to your RDS or Aurora MySQL instance. For more information about VPC networking, see the following.

- [What is VPC peering?](#) in the *Amazon VPC Peering Guide*
- [Working with a DB instance in a VPC](#) in the *Amazon RDS User Guide*

Note

If your Amazon Redshift cluster is in a different VPC than your RDS or Aurora MySQL instance, then enable enhanced VPC routing. Otherwise, you might receive timeout errors when you run a federated query.

2. Set up secrets in AWS Secrets Manager for your RDS MySQL and Aurora MySQL databases. Then reference the secrets in AWS Identity and Access Management (IAM) access policies and roles. For more information, see [Creating a secret and an IAM role to use federated queries \(p. 238\)](#).

Note

If your cluster uses enhanced VPC routing, you might need to configure an interface VPC endpoint for AWS Secrets Manager. This is necessary when the VPC and subnet of your Amazon Redshift cluster don't have access to the public AWS Secrets Manager endpoint. When you use a VPC interface endpoint, communication between the Amazon Redshift cluster in your VPC and AWS Secrets Manager is routed privately from your VPC to the endpoint interface. For more information, see [Creating an interface endpoint](#) in the *Amazon VPC User Guide*.

3. Apply the IAM role that you previously created to the Amazon Redshift cluster. For more information, see [Creating a secret and an IAM role to use federated queries \(p. 238\)](#).
4. Connect to your RDS MySQL and Aurora MySQL databases with an external schema. For more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#). For examples on how to use federated queries, see [Example of using a federated query with MySQL \(p. 243\)](#).
5. Run your SQL queries referencing the external schema that references your RDS MySQL and Aurora MySQL databases.

Creating a secret and an IAM role to use federated queries

The following steps show how to create a secret and an IAM role to use with federated queries.

Prerequisites

Make sure that you have the following prerequisites to create a secret and an IAM role to use with federated queries:

- An RDS PostgreSQL, Aurora PostgreSQL DB instance, RDS MySQL, or Aurora MySQL DB instance with user name and password authentication.
- An Amazon Redshift cluster with a cluster maintenance version that supports federated queries.

To create a secret (user name and password) with AWS Secrets Manager

1. Sign in to the Secrets Manager console with the account that owns your RDS or Aurora instance.
2. Choose **Store a new secret**.
3. Choose the **Credentials for RDS database** tile. For **User name** and **Password**, enter values for your instance. Confirm or choose a value for **Encryption key**. Then choose the RDS database that your secret will access.

Note

We recommend using the default encryption key (DefaultEncryptionKey). If you use a custom encryption key, the IAM role that is used to access the secret must be added as a key user.

4. Enter a name for the secret, continue with the creation steps with the default choices, and then choose **Store**.
5. View your secret and note the **Secret ARN** value that you created to identify the secret.

To create a security policy using the secret

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Create a policy with JSON similar to the following.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AccessSecret",  
            "Effect": "Allow",  
            "Action": [  
                "secretsmanager:GetResourcePolicy",  
                "secretsmanager:GetSecretValue",  
                "secretsmanager:DescribeSecret",  
                "secretsmanager>ListSecretVersionIds"  
            ],  
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-rds-secret-VNenFy"  
        },  
        {  
            "Sid": "VisualEditor1",  
            "Effect": "Allow",  
            "Action": [  
                "secretsmanager:GetRandomPassword",  
                "secretsmanager>ListSecrets"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

To retrieve the secret, you need list and read actions. We recommend that you restrict the resource to the specific secret that you created. To do this, use the Amazon Resource Name (ARN) of the secret to limit the resource. You can also specify the permissions and resources using the visual editor on the IAM console.

3. Give the policy a name and finish creating it.
4. Navigate to **IAM roles**.
5. Create an IAM role for **Redshift - Customizable**.
6. Either attach the IAM policy you just created to an existing IAM role, or create a new IAM role and attach the policy.
7. On the **Trust relationships** tab of your IAM role, confirm that the role contains the trust entity `redshift.amazonaws.com`.
8. Note the **Role ARN** you created. This ARN has access to the secret.

To attach the IAM role to your Amazon Redshift cluster

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Clusters**. The clusters for your account in the current AWS Region are listed.
3. Choose the cluster name in the list to view more details about a cluster.
4. For **Actions**, choose **Manage IAM roles**. The **Manage IAM roles** page appears.
5. Add your IAM role to the cluster.

Examples of using a federated query

The following examples show how to run a federated query.

Example of using a federated query with PostgreSQL

The following example shows how to set up a federated query that references an Amazon Redshift database, an Aurora PostgreSQL database, and Amazon S3. This example illustrates how federated queries work. To run it on your own environment, change it to fit your environment. For prerequisites for doing this, see [Getting started with using federated queries to PostgreSQL \(p. 234\)](#).

Create an external schema that references an Aurora PostgreSQL database.

```
CREATE EXTERNAL SCHEMA apg
FROM POSTGRES
DATABASE 'database-1' SCHEMA 'myschema'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-R0'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/dataplane-
apg-creds-YbVKQw';
```

Create another external schema that references Amazon S3, which uses Amazon Redshift Spectrum. Also, grant permission to use the schema to public.

```
CREATE EXTERNAL SCHEMA s3
FROM DATA CATALOG
DATABASE 'default' REGION 'us-west-2'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-S3';

GRANT USAGE ON SCHEMA s3 TO public;
```

Show the count of rows in the Amazon Redshift table.

```
SELECT count(*) FROM public.lineitem;

count
-----
25075099
```

Show the count of rows in the Aurora PostgreSQL table.

```
SELECT count(*) FROM apg.lineitem;
```

```
count
-----
11760
```

Show the count of rows in Amazon S3.

```
SELECT count(*) FROM s3.lineitem_1t_part;
count
-----
6144008876
```

Create a view of the tables from Amazon Redshift, Aurora PostgreSQL, and Amazon S3. This view is used to run your federated query.

```
CREATE VIEW lineitem_all AS
  SELECT
    l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity, l_extendedprice, l_discount, l_tax, l_returnflag, l
    l_shipdate::date, l_commitdate::date, l_receiptdate::date,
    l_shipinstruct, l_shipmode, l_comment
   FROM s3.lineitem_1t_part
 UNION ALL SELECT * FROM public.lineitem
 UNION ALL SELECT * FROM apg.lineitem
   with no schema binding;
```

Show the count of rows in the view lineitem_all with a predicate to limit the results.

```
SELECT count(*) from lineitem_all WHERE l_quantity = 10;
count
-----
123373836
```

Find out how many sales of one item there were in January of each year.

```
SELECT extract(year from l_shipdate) as year,
       extract(month from l_shipdate) as month,
       count(*) as orders
  FROM lineitem_all
 WHERE extract(month from l_shipdate) = 1
   AND l_quantity < 2
 GROUP BY 1,2
 ORDER BY 1,2;

  year | month | orders
-----+-----+
  1992 |     1 | 196019
  1993 |     1 | 1582034
  1994 |     1 | 1583181
  1995 |     1 | 1583919
  1996 |     1 | 1583622
  1997 |     1 | 1586541
  1998 |     1 | 1583198
  2016 |     1 | 15542
  2017 |     1 | 15414
  2018 |     1 | 15527
  2019 |     1 |      151
```

Example of using a mixed-case name

To query a supported PostgreSQL remote database that has a mixed-case name of a database, schema, table, or column, then set `enable_case_sensitive_identifier` to true. For more information about this session parameter, see [enable_case_sensitive_identifier \(p. 1688\)](#).

```
SET enable_case_sensitive_identifier TO TRUE;
```

Typically, the database and schema names are in lowercase. The following example shows how you can connect to a supported PostgreSQL remote database that has lowercase names for database and schema and mixed-case names for table and column.

Create an external schema that references an Aurora PostgreSQL database that has a lowercase database name (`dblower`) and lowercase schema name (`schemalower`).

```
CREATE EXTERNAL SCHEMA apg_lower
FROM POSTGRES
DATABASE 'dblower' SCHEMA 'schemalower'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/dataplane-
apg-creds-YbVKQw';
```

In the session where the query runs, set `enable_case_sensitive_identifier` to true.

```
SET enable_case_sensitive_identifier TO TRUE;
```

Run a federated query to select all data from the PostgreSQL database. The table (`MixedCaseTab`) and column (`MixedCaseName`) have mixed-case names. The result is one row (`Harry`).

```
select * from apg_lower."MixedCaseTab";
```

```
MixedCaseName
-----
Harry
```

The following example shows how you can connect to a supported PostgreSQL remote database that has a mixed-case name for the database, schema, table, and column.

Set `enable_case_sensitive_identifier` to true before you create the external schema. If `enable_case_sensitive_identifier` is not set to true before creating the external schema, then a database does not exist error occurs.

Create an external schema that references an Aurora PostgreSQL database that has a mixed-case database (`UpperDB`) and schema (`UpperSchema`) name.

```
CREATE EXTERNAL SCHEMA apg_upper
FROM POSTGRES
DATABASE 'UpperDB' SCHEMA 'UpperSchema'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/dataplane-
apg-creds-YbVKQw';
```

Run a federated query to select all data from the PostgreSQL database. The table (`MixedCaseTab`) and column (`MixedCaseName`) have mixed-case names. The result is one row (`Harry`).

```
select * from apg_upper."MixedCaseTab";
```

```
MixedCaseName
-----
Harry
```

Example of using a federated query with MySQL

The following example shows how to set up a federated query that references an Aurora MySQL database. This example illustrates how federated queries works. To run it on your own environment, change it to fit your environment. For prerequisites for doing this, see [Getting started with using federated queries to MySQL \(p. 237\)](#).

This example depends on the following prerequisites:

- A secret that was set up in Secrets Manager for the Aurora MySQL database. This secret is referenced in IAM access policies and roles. For more information, see [Creating a secret and an IAM role to use federated queries \(p. 238\)](#).
- A security group that is set up linking Amazon Redshift and Aurora MySQL.

Create an external schema that references an Aurora MySQL database.

```
CREATE EXTERNAL SCHEMA amysql
FROM MYSQL
DATABASE 'functional'
URI 'endpoint to remote hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/dataplane-
apg-creds-YbVKQw';
```

Run an example SQL select of the Aurora MySQL table to display one row from the employees table in Aurora MySQL.

```
SELECT level FROM amysql.employees LIMIT 1;
level
-----
8
```

Data type differences between Amazon Redshift and supported PostgreSQL and MySQL databases

The following table shows the mapping of an Amazon Redshift data type to a corresponding Amazon RDS PostgreSQL or Aurora PostgreSQL data type.

Amazon Redshift data type	RDS PostgreSQL or Aurora PostgreSQL data type	Description
SMALLINT	SMALLINT	Signed two-byte integer

Amazon Redshift data type	RDS PostgreSQL or Aurora PostgreSQL data type	Description
INTEGER	INTEGER	Signed four-byte integer
BIGINT	BIGINT	Signed eight-byte integer
DECIMAL	DECIMAL	Exact numeric of selectable precision
REAL	REAL	Single precision floating-point number
DOUBLE PRECISION	DOUBLE PRECISION	Double precision floating-point number
BOOLEAN	BOOLEAN	Logical Boolean (true/false)
CHAR	CHAR	Fixed-length character string
VARCHAR	VARCHAR	Variable-length character string with a user-defined limit
DATE	DATE	Calendar date (year, month, day)
TIMESTAMP	TIMESTAMP	Date and time (without time zone)
TIMESTAMPTZ	TIMESTAMPTZ	Date and time (with time zone)
GEOMETRY	PostGIS GEOMETRY	Spatial data

The following RDS PostgreSQL and Aurora PostgreSQL data types are converted to VARCHAR(64K) in Amazon Redshift:

- JSON, JSONB
- Arrays
- BIT, BIT VARYING
- BYTEA
- Composite types
- Date and time types INTERVAL, TIME, TIME WITH TIMEZONE
- Enumerated types
- Monetary types
- Network address types
- Numeric types SERIAL, BIGSERIAL, SMALLSERIAL, and MONEY
- Object identifier types
- pg_lsn type
- Pseudotypes

- Range types
- Text search types
- TXID_SNAPSHOT
- UUID
- XML type

The following table shows the mapping of an Amazon Redshift data type to a corresponding Amazon RDS MySQL or Aurora MySQL data type.

Amazon Redshift data type	RDS MySQL or Aurora MySQL data type	Description
BOOLEAN	TINYINT(1)	Logical Boolean (true or false)
SMALLINT	TINYINT(UNSIGNED)	Signed two-byte integer
SMALLINT	SMALLINT	Signed two-byte integer
INTEGER	SMALLINT UNSIGNED	Signed four-byte integer
INTEGER	MEDIUMINT (UNSIGNED)	Signed four-byte integer
INTEGER	INT	Signed four-byte integer
BIGINT	INT UNSIGNED	Signed eight-byte integer
BIGINT	BIGINT	Signed eight-byte integer
DECIMAL	BIGINT UNSIGNED	Exact numeric of selectable precision
DECIMAL	DECIMAL(M,D)	Exact numeric of selectable precision
REAL	FLOAT	Single precision floating-point number
DOUBLE PRECISION	DOUBLE	Double precision floating-point number
CHAR	CHAR	Fixed-length character string
VARCHAR	VARCHAR	Variable-length character string with a user-defined limit
DATE	DATE	Calendar date (year, month, day)

Amazon Redshift data type	RDS MySQL or Aurora MySQL data type	Description
TIME	TIME	Time (without time zone)
TIMESTAMP	TIMESTAMP	Date and time (without time zone)
TIMESTAMP	DATETIME	Time (without time zone)
VARCHAR(4)	YEAR	Variable length character representing year

An error results when TIME data is out of range (00:00:00 – 24:00:00).

The following RDS MySQL and Aurora MySQL data types are converted to VARCHAR(64K) in Amazon Redshift:

- BIT
- BINARY
- VARBINARY
- TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB
- TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT
- ENUM
- SET
- SPATIAL

Considerations when accessing federated data with Amazon Redshift

Some Amazon Redshift features don't support access to federated data. You can find related limitations and considerations following.

The following are limitations and considerations when using federated queries with Amazon Redshift:

- Federated queries support read access to external data sources. You can't write or create database objects in the external data source.
- In some cases, you might access an Amazon RDS or Aurora database in a different AWS Region than Amazon Redshift. In these cases, you typically incur network latency and billing charges for transferring data across AWS Regions. We recommend using an Aurora global database with a local endpoint in the same AWS Region as your Amazon Redshift cluster. Aurora global databases use dedicated infrastructure for storage-based replication across any two AWS Regions with typical latency of less than 1 second.
- Consider the cost of accessing Amazon RDS or Aurora. For example, when using this feature to access Aurora, Aurora charges are based on IOPS.
- Federated queries don't enable access to Amazon Redshift from RDS or Aurora.
- Federated queries are only available in AWS Regions where both Amazon Redshift and Amazon RDS or Aurora are available.

- Federated queries currently don't support ALTER SCHEMA. To change a schema, use DROP and then CREATE EXTERNAL SCHEMA.
- Federated queries don't work with concurrency scaling.
- Federated queries currently don't support access through a PostgreSQL foreign data wrapper.
- Federated queries to RDS MySQL or Aurora MySQL support transaction isolation at the READ COMMITTED level.
- If not specified, Amazon Redshift connects to RDS MySQL or Aurora MySQL on port 3306. Confirm the MySQL port number before creating an external schema for MySQL.
- When fetching TIMESTAMP and DATE data types from MySQL, zero values are treated as NULL.
- If an Aurora database reader endpoint is used, an "invalid snapshot" error can occur. This can be avoided by one of the following methods:
 - Use a specific Aurora instance endpoint (instead of using the Aurora cluster endpoint). This method uses REPEATABLE READ transaction isolation for the results from the PostgreSQL database.
 - Use an Aurora reader endpoint and set pg_federation_repeatable_read to false for the session. This method uses READ COMMITTED transaction isolation for the results from the PostgreSQL database. For more information about Aurora reader endpoints, see [Types of Aurora endpoints](#) in the *Amazon Aurora User Guide*. For information about pg_federation_repeatable_read, see [pg_federation_repeatable_read \(p. 1695\)](#).

The following are considerations for transactions when working with federated queries to PostgreSQL databases:

- If a query consists of federated tables, the leader node starts a READ ONLY REPEATABLE READ transaction on the remote database. This transaction remains for the duration of the Amazon Redshift transaction.
- The leader node creates a snapshot of the remote database by calling pg_export_snapshot and makes a read lock on the affected tables.
- A compute node starts a transaction and uses the snapshot created at the leader node to issue queries to the remote database.

An Amazon Redshift external schema can reference a database in an external RDS PostgreSQL or Aurora PostgreSQL. When it does, these limitations apply:

- When creating an external schema referencing Aurora, the Aurora PostgreSQL database must be at version 9.6, or later.
- When creating an external schema referencing Amazon RDS, the Amazon RDS PostgreSQL database must be at version 9.6, or later.

An Amazon Redshift external schema can reference a database in an external RDS MySQL or Aurora MySQL. When it does, these limitations apply:

- When creating an external schema referencing Aurora, the Aurora MySQL database must be at version 5.6 or later.
- When creating an external schema referencing Amazon RDS, the RDS MySQL database must be at version 5.6 or later.

Querying external data using Amazon Redshift Spectrum

Using Amazon Redshift Spectrum, you can efficiently query and retrieve structured and semistructured data from files in Amazon S3 without having to load the data into Amazon Redshift tables. Redshift Spectrum queries employ massive parallelism to run very fast against large datasets. Much of the processing occurs in the Redshift Spectrum layer, and most of the data remains in Amazon S3. Multiple clusters can concurrently query the same dataset in Amazon S3 without the need to make copies of the data for each cluster.

Topics

- [Amazon Redshift Spectrum overview \(p. 248\)](#)
- [Getting started with Amazon Redshift Spectrum \(p. 250\)](#)
- [IAM policies for Amazon Redshift Spectrum \(p. 259\)](#)
- [Using Redshift Spectrum with AWS Lake Formation \(p. 269\)](#)
- [Creating data files for queries in Amazon Redshift Spectrum \(p. 270\)](#)
- [Creating external schemas for Amazon Redshift Spectrum \(p. 272\)](#)
- [Creating external tables for Redshift Spectrum \(p. 277\)](#)
- [Improving Amazon Redshift Spectrum query performance \(p. 287\)](#)
- [Setting data handling options \(p. 289\)](#)
- [Example: Performing correlated subqueries in Redshift Spectrum \(p. 290\)](#)
- [Monitoring metrics in Amazon Redshift Spectrum \(p. 290\)](#)
- [Troubleshooting queries in Amazon Redshift Spectrum \(p. 291\)](#)
- [Tutorial: Querying nested data with Amazon Redshift Spectrum \(p. 294\)](#)

Amazon Redshift Spectrum overview

Amazon Redshift Spectrum resides on dedicated Amazon Redshift servers that are independent of your cluster. Amazon Redshift pushes many compute-intensive tasks, such as predicate filtering and aggregation, down to the Redshift Spectrum layer. Thus, Redshift Spectrum queries use much less of your cluster's processing capacity than other queries. Redshift Spectrum also scales intelligently. Based on the demands of your queries, Redshift Spectrum can potentially use thousands of instances to take advantage of massively parallel processing.

You create Redshift Spectrum tables by defining the structure for your files and registering them as tables in an external data catalog. The external data catalog can be AWS Glue, the data catalog that comes with Amazon Athena, or your own Apache Hive metastore. You can create and manage external tables either from Amazon Redshift using data definition language (DDL) commands or using any other tool that connects to the external data catalog. Changes to the external data catalog are immediately available to any of your Amazon Redshift clusters.

Optionally, you can partition the external tables on one or more columns. Defining partitions as part of the external table can improve performance. The improvement occurs because the Amazon Redshift query optimizer eliminates partitions that don't contain data for the query.

After your Redshift Spectrum tables have been defined, you can query and join the tables just as you do any other Amazon Redshift table. Redshift Spectrum doesn't support update operations on external

tables. You can add Redshift Spectrum tables to multiple Amazon Redshift clusters and query the same data on Amazon S3 from any cluster in the same AWS Region. When you update Amazon S3 data files, the data is immediately available for query from any of your Amazon Redshift clusters.

The AWS Glue Data Catalog that you access might be encrypted to increase security. If the AWS Glue catalog is encrypted, you need the AWS Key Management Service (AWS KMS) key for AWS Glue to access the AWS Glue catalog. AWS Glue catalog encryption is not available in all AWS Regions. For a list of supported AWS Regions, see [Encryption and Secure Access for AWS Glue](#) in the [AWS Glue Developer Guide](#). For more information about AWS Glue Data Catalog encryption, see [Encrypting Your AWS Glue Data Catalog](#) in the [AWS Glue Developer Guide](#).

Note

You can't view details for Redshift Spectrum tables using the same resources that you use for standard Amazon Redshift tables, such as [PG_TABLE_DEF \(p. 1677\)](#), [STV_TBL_PERM \(p. 1599\)](#), [PG_CLASS](#), or [information_schema](#). If your business intelligence or analytics tool doesn't recognize Redshift Spectrum external tables, configure your application to query [SVV_EXTERNAL_TABLES \(p. 1431\)](#) and [SVV_EXTERNAL_COLUMNS \(p. 1428\)](#).

Amazon Redshift Spectrum Regions

Redshift Spectrum is available in AWS Regions where Amazon Redshift is available, unless otherwise specified in Region specific documentation. For AWS Region availability in commercial Regions, see [Service endpoints](#) for the [Redshift API](#) in the [Amazon Web Services General Reference](#).

Amazon Redshift Spectrum considerations

Note the following considerations when you use Amazon Redshift Spectrum:

- The Amazon Redshift cluster and the Amazon S3 bucket must be in the same AWS Region.
- Redshift Spectrum doesn't support enhanced VPC routing with provisioned clusters. To access your Amazon S3 data, you might need to perform additional configuration steps. For more information, see [Redshift Spectrum and enhanced VPC routing](#) in the [Amazon Redshift Management Guide](#).
- Redshift Spectrum supports Amazon S3 access point aliases. For more information, see [Using a bucket-style alias for your access point](#) in the [Amazon Simple Storage Service User Guide](#). However, Redshift Spectrum doesn't support VPC with Amazon S3 access point aliases. For more information, see [Redshift Spectrum and enhanced VPC routing](#) in the [Amazon Redshift Management Guide](#).
- You can't perform update or delete operations on external tables. To create a new external table in the specified schema, you can use CREATE EXTERNAL TABLE. For more information about CREATE EXTERNAL TABLE, see [CREATE EXTERNAL TABLE \(p. 768\)](#). To insert the results of a SELECT query into existing external tables on external catalogs, you can use INSERT (external table). For more information about INSERT (external table), see [INSERT \(external table\) \(p. 906\)](#).
- Unless you are using an AWS Glue Data Catalog that is enabled for AWS Lake Formation, you can't control user permissions on an external table. Instead, you can grant and revoke permissions on the external schema. For more information about working with AWS Lake Formation, see [Using Redshift Spectrum with AWS Lake Formation \(p. 269\)](#).
- To run Redshift Spectrum queries, the database user must have permission to create temporary tables in the database. The following example grants temporary permission on the database spectrumdb to the spectrumusers user group.

```
grant temp on database spectrumdb to group spectrumusers;
```

For more information, see [GRANT \(p. 887\)](#).

- When using the Athena Data Catalog or AWS Glue Data Catalog as a metadata store, see [Quotas and Limits](#) in the [Amazon Redshift Management Guide](#).

- Redshift Spectrum doesn't support Amazon EMR with Kerberos.

Getting started with Amazon Redshift Spectrum

In this tutorial, you learn how to use Amazon Redshift Spectrum to query data directly from files on Amazon S3. If you already have a cluster and a SQL client, you can complete this tutorial with minimal setup.

Note

Redshift Spectrum queries incur additional charges. The cost of running the sample queries in this tutorial is nominal. For more information about pricing, see [Amazon Redshift Spectrum pricing](#).

Prerequisites

To use Redshift Spectrum, you need an Amazon Redshift cluster and a SQL client that's connected to your cluster so that you can run SQL commands. The cluster and the data files in Amazon S3 must be in the same AWS Region.

For information about how to create an Amazon Redshift cluster, see [Amazon Redshift clusters and data loading](#) in the *Amazon Redshift Getting Started Guide*. For information about ways to connect to a cluster, see [Connecting to Amazon Redshift provisioned clusters](#) in the *Amazon Redshift Getting Started Guide*.

In some of the examples that follow, the sample data is in the US East (N. Virginia) Region (us-east-1), so you need a cluster that is also in us-east-1. Or, you can use Amazon S3 to copy data objects from the following buckets and folders to your bucket in the AWS Region where your cluster is located:

- s3://redshift-downloads/ticket/spectrum/customers/*
- s3://redshift-downloads/ticket/spectrum/sales_partition/*
- s3://redshift-downloads/ticket/spectrum/sales/*
- s3://redshift-downloads/ticket/spectrum/salesevent/*

Run an Amazon S3 command similar to the following to copy sample data that is located in the US East (N. Virginia) to your AWS Region. Before running the command create your bucket and folders in your bucket to match your Amazon S3 copy command. The output of the Amazon S3 copy command confirms that the files are copied to the *bucket-name* in your desired AWS Region.

```
aws s3 cp s3://redshift-downloads/ticket/spectrum/ s3://bucket-name/ticket/spectrum/ --copy-props none --recursive
```

Getting started with Redshift Spectrum using AWS CloudFormation

As an alternative to the following steps, you can access the Redshift Spectrum DataLake AWS CloudFormation template to create a stack with an Amazon S3 bucket that you can query. For more information, see [Launch your AWS CloudFormation stack and then query your data in Amazon S3 \(p. 257\)](#).

Getting started with Redshift Spectrum step by step

To get started using Amazon Redshift Spectrum, follow these steps:

- [Step 1. Create an IAM role for Amazon Redshift \(p. 251\)](#)
- [Step 2: Associate the IAM role with your cluster \(p. 253\)](#)
- [Step 3: Create an external schema and an external table \(p. 254\)](#)
- [Step 4: Query your data in Amazon S3 \(p. 255\)](#)

Step 1. Create an IAM role for Amazon Redshift

Your cluster needs authorization to access your external Data Catalog in AWS Glue or Amazon Athena and your data files in Amazon S3. To provide that authorization, you reference an AWS Identity and Access Management (IAM) role that is attached to your cluster. For more information about using roles with Amazon Redshift, see [Authorizing COPY and UNLOAD Operations Using IAM Roles](#).

Note

In certain cases, you can migrate your Athena Data Catalog to an AWS Glue Data Catalog. You can do this if your cluster is in an AWS Region where AWS Glue is supported and you have Redshift Spectrum external tables in the Athena Data Catalog. To use the AWS Glue Data Catalog with Redshift Spectrum, you might need to change your IAM policies. For more information, see [Upgrading to the AWS Glue Data Catalog](#) in the *Athena User Guide*.

When you create a role for Amazon Redshift, choose one of the following approaches:

- If you are using Redshift Spectrum with either an Athena Data Catalog or AWS Glue Data Catalog, follow the steps outlined in [To create an IAM role for Amazon Redshift \(p. 251\)](#).
- If you are using Redshift Spectrum with an AWS Glue Data Catalog that is enabled for AWS Lake Formation, follow the steps outlined in these procedures:
 - [To create an IAM role for Amazon Redshift using an AWS Glue Data Catalog enabled for AWS Lake Formation \(p. 252\)](#)
 - [To grant SELECT permissions on the table to query in the Lake Formation database \(p. 253\)](#)

To create an IAM role for Amazon Redshift

1. Open the [IAM console](#).
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. Choose **AWS service** as the trusted entity, and then choose **Redshift** as the use case.
5. Under **Use case for other AWS services**, choose **Redshift - Customizable** and then choose **Next**.
6. The **Add permissions policy** page appears. Choose **AmazonS3ReadOnlyAccess** and **AWSGlueConsoleFullAccess**, if you're using the AWS Glue Data Catalog. Or choose **AmazonAthenaFullAccess** if you're using the Athena Data Catalog. Choose **Next**.

Note

The **AmazonS3ReadOnlyAccess** policy gives your cluster read-only access to all Amazon S3 buckets. To grant access to only the AWS sample data bucket, create a new policy and add the following permissions.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:Get*",  
                "s3>List*"
```

```
        ],
        "Resource": "arn:aws:s3:::redshift-downloads/*"
    }
}
```

7. For **Role name**, enter a name for your role, for example **myspectrum_role**.
8. Review the information, and then choose **Create role**.
9. In the navigation pane, choose **Roles**. Choose the name of your new role to view the summary, and then copy the **Role ARN** to your clipboard. This value is the Amazon Resource Name (ARN) for the role that you just created. You use that value when you create external tables to reference your data files on Amazon S3.

To create an IAM role for Amazon Redshift using an AWS Glue Data Catalog enabled for AWS Lake Formation

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**.

If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.

3. Choose **Create policy**.
4. Choose to create the policy on the **JSON** tab.
5. Paste in the following JSON policy document, which grants access to the Data Catalog but denies the administrator permissions for Lake Formation.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RedshiftPolicyForLF",
            "Effect": "Allow",
            "Action": [
                "glue:*",
                "lakeformation:GetDataAccess"
            ],
            "Resource": "*"
        }
    ]
}
```

6. When you are finished, choose **Review** to review the policy. The policy validator reports any syntax errors.
7. On the **Review policy** page, for **Name** enter **myspectrum_policy** to name the policy that you are creating. Enter a **Description** (optional). Review the policy **Summary** to see the permissions that are granted by your policy. Then choose **Create policy** to save your work.

After you create a policy, you can provide access to your users.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center (successor to AWS Single Sign-On):

Create a permission set. Follow the instructions in [Create a permission set](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

- Users managed in IAM through an identity provider:

Create a role for identity federation. Follow the instructions in [Creating a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*.

- IAM users:
 - Create a role that your user can assume. Follow the instructions in [Creating a role for an IAM user](#) in the *IAM User Guide*.
 - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in [Adding permissions to a user \(console\)](#) in the *IAM User Guide*.

To grant SELECT permissions on the table to query in the Lake Formation database

1. Open the Lake Formation console at <https://console.aws.amazon.com/lakeformation/>.
 2. In the navigation pane, choose **Permissions**, and then choose **Grant**.
 3. Provide the following information:
 - For **IAM role**, choose the IAM role you created, `myspectrum_role`. When you run the Amazon Redshift Query Editor, it uses this IAM role for permission to the data.
- Note**
To grant SELECT permission on the table in a Lake Formation–enabled Data Catalog to query, do the following:
- Register the path for the data in Lake Formation.
 - Grant users permission to that path in Lake Formation.
 - Created tables can be found in the path registered in Lake Formation.
- For **Database**, choose your Lake Formation database.
 - For **Table**, choose a table within the database to query.
 - For **Columns**, choose **All Columns**.
 - Choose the **Select** permission.
4. Choose **Save**.

Important

As a best practice, allow access only to the underlying Amazon S3 objects through Lake Formation permissions. To prevent unapproved access, remove any permission granted to Amazon S3 objects outside of Lake Formation. If you previously accessed Amazon S3 objects before setting up Lake Formation, remove any IAM policies or bucket permissions that previously were set up. For more information, see [Upgrading AWS Glue Data Permissions to the AWS Lake Formation Model](#) and [Lake Formation Permissions](#).

Step 2: Associate the IAM role with your cluster

Now you have an IAM role that authorizes Amazon Redshift to access the external Data Catalog and Amazon S3 for you. At this point, you must associate that role with your Amazon Redshift cluster.

To associate an IAM role with a cluster

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Clusters**, then choose the name of the cluster that you want to update.
3. For **Actions**, choose **Manage IAM roles**. The **IAM roles** page appears.
4. Either choose **Enter ARN** and then enter an ARN or an IAM role, or choose an IAM role from the list. Then choose **Add IAM role** to add it to the list of **Attached IAM roles**.

5. Choose **Done** to associate the IAM role with the cluster. The cluster is modified to complete the change.

Step 3: Create an external schema and an external table

Create external tables in an external schema. The external schema references a database in the external data catalog and provides the IAM role ARN that authorizes your cluster to access Amazon S3 on your behalf. You can create an external database in an Amazon Athena Data Catalog, AWS Glue Data Catalog, or an Apache Hive metastore, such as Amazon EMR. For this example, you create the external database in an Amazon Athena Data Catalog when you create the external schema Amazon Redshift. For more information, see [Creating external schemas for Amazon Redshift Spectrum \(p. 272\)](#).

To create an external schema and an external table

1. To create an external schema, replace the IAM role ARN in the following command with the role ARN you created in [step 1 \(p. 251\)](#). Then run the command in your SQL client.

```
create external schema myspectrum_schema
from data catalog
database 'myspectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myspectrum_role'
create external database if not exists;
```

2. To create an external table, run the following CREATE EXTERNAL TABLE command.

Note

Your cluster and the Amazon S3 bucket must be in the same AWS Region. For this example CREATE EXTERNAL TABLE command, the Amazon S3 bucket with the sample data is located in the US East (N. Virginia) AWS Region. To see the source data, download the [sales_ts.000 file](#).

You can modify this example to run in a different AWS Region. Create an Amazon S3 bucket in your desired AWS Region. Copy the sales data with an Amazon S3 copy command. Then update the location option in the example CREATE EXTERNAL TABLE command to your bucket.

```
aws s3 cp s3://redshift-downloads/ticket/spectrum/sales/ s3://bucket-name/
ticket/spectrum/sales/ --copy-props none --recursive
```

The output of the Amazon S3 copy command confirms that the file was copied to the *bucket-name* in your desired AWS Region.

```
copy: s3://redshift-downloads/ticket/spectrum/sales/sales_ts.000 to
s3://bucket-name/ticket/spectrum/sales/sales_ts.000
```

```
create external table myspectrum_schema.sales(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtypaid smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
```

```
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://redshift-downloads/ticket/spectrum/sales/'
table properties ('numRows'='172000');
```

Step 4: Query your data in Amazon S3

After your external tables are created, you can query them using the same SELECT statements that you use to query other Amazon Redshift tables. These SELECT statement queries include joining tables, aggregating data, and filtering on predicates.

To query your data in Amazon S3

1. Get the number of rows in the MYSPECTRUM_SCHEMA.SALES table.

```
select count(*) from myspectrum_schema.sales;
```

```
count
-----
172462
```

2. Keep your larger fact tables in Amazon S3 and your smaller dimension tables in Amazon Redshift, as a best practice. If you loaded the sample data in [Getting Started with Amazon Redshift](#), you have a table named EVENT in your database. If not, create the EVENT table by using the following command.

```
create table event(
eventid integer not null distkey,
venueid smallint not null,
catid smallint not null,
dateid smallint not null sortkey,
eventname varchar(200),
starttime timestamp);
```

3. Load the EVENT table by replacing the IAM role ARN in the following COPY command with the role ARN you created in [Step 1. Create an IAM role for Amazon Redshift \(p. 251\)](#). You can optionally download and view the [source data for the allevents_pipe.txt](#) from an Amazon S3 bucket in AWS Region us-east-1.

```
copy event from 's3://redshift-downloads/ticket/allevents_pipe.txt'
iam_role 'arn:aws:iam::123456789012:role/myspectrum_role'
delimiter '|' timeformat 'YYYY-MM-DD HH:MI:SS' region 'us-east-1';
```

The following example joins the external Amazon S3 table MYSPECTRUM_SCHEMA.SALES with the local Amazon Redshift table EVENT to find the total sales for the top 10 events.

```
select top 10 myspectrum_schema.sales.eventid, sum(myspectrum_schema.sales.pricepaid)
  from myspectrum_schema.sales, event
 where myspectrum_schema.sales.eventid = event.eventid
 and myspectrum_schema.sales.pricepaid > 30
 group by myspectrum_schema.sales.eventid
 order by 2 desc;
```

```
eventid | sum
-----+-----
```

289		51846.00
7895		51049.00
1602		50301.00
851		49956.00
7315		49823.00
6471		47997.00
2118		47863.00
984		46780.00
7851		46661.00
5638		46280.00

- View the query plan for the previous query. Notice the S3 Seq Scan, S3 HashAggregate, and S3 Query Scan steps that were run against the data on Amazon S3.

```
explain
select top 10 myspectrum_schema.sales.eventid, sum(myspectrum_schema.sales.pricepaid)
from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

QUERY PLAN

```
-----
XN Limit  (cost=1001055770628.63..1001055770628.65 rows=10 width=31)

-> XN Merge  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

    Merge Key: sum(sales.derived_col2)

    -> XN Network  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

        Send to leader

        -> XN Sort  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

            Sort Key: sum(sales.derived_col2)

            -> XN HashAggregate  (cost=1055770620.49..1055770620.99 rows=200
width=31)

                -> XN Hash Join DS_BCAST_INNER  (cost=3119.97..1055769620.49
rows=200000 width=31)

                    Hash Cond: ("outer".derived_col1 = "inner".eventid)

                    -> XN S3 Query Scan sales  (cost=3010.00..5010.50
rows=200000 width=31)

                    -> S3 HashAggregate  (cost=3010.00..3010.50
rows=200000 width=16)
```

```
-> S3 Seq Scan myspectrum_schema.sales
location:"s3://redshift-downloads/ticket/spectrum/sales" format:TEXT
(cost=0.00..2150.00 rows=172000 width=16)
Filter: (pricepaid > 30.00)

-> XN Hash (cost=87.98..87.98 rows=8798 width=4)

-> XN Seq Scan on event (cost=0.00..87.98
rows=8798 width=4)
```

Launch your AWS CloudFormation stack and then query your data in Amazon S3

After you create an Amazon Redshift cluster and connect to the cluster, you can install your Redshift Spectrum DataLake AWS CloudFormation template and then query your data.

CloudFormation installs the Redshift Spectrum Getting Started DataLake template and creates a stack that includes the following:

- A role named `myspectrum_role` associated with your Redshift cluster
- An external schema named `myspectrum_schema`
- An external table named `sales` in an Amazon S3 bucket
- A Redshift table named `event` loaded with data

To launch your Redshift Spectrum Getting Started DataLake CloudFormation stack

1. Choose [Launch CFN stack](#). The CloudFormation console opens with the `DataLake.yml` template selected.

You can also download and customize the Redshift Spectrum Getting Started DataLake CloudFormation [CFN template](#), then open CloudFormation console (<https://console.aws.amazon.com/cloudformation>) and create a stack with the customized template.

2. Choose **Next**.
3. Under **Parameters**, enter the Amazon Redshift cluster name, database name, and your database user name.
4. Choose **Next**.

The stack options appear.

5. Choose **Next** to accept the default settings.
6. Review the information and under **Capabilities**, and choose **I acknowledge that AWS CloudFormation might create IAM resources**.
7. Choose **Create stack**.

If an error occurs while the stack is being created, see the following information:

- View the CloudFormation **Events** tab for information that can help you resolve the error.
- Delete the DataLake CloudFormation stack before trying the operation again.
- Make sure that you are connected to your Amazon Redshift database.
- Make sure that you entered the correct information for the Amazon Redshift cluster name, database name, and database user name.

Querying your data in Amazon S3

You query external tables using the same SELECT statements that you use to query other Amazon Redshift tables. These SELECT statement queries include joining tables, aggregating data, and filtering on predicates.

The following query returns the number of rows in the myspectrum_schema.sales external table.

```
select count(*) from myspectrum_schema.sales;
```

```
count
-----
172462
```

Joining an external table with a local table

The following example joins the external table myspectrum_schema.sales with the local table event to find the total sales for the top 10 events.

```
select top 10 myspectrum_schema.sales.eventid, sum(myspectrum_schema.sales.pricepaid) from
myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

eventid	sum
289	51846.00
7895	51049.00
1602	50301.00
851	49956.00
7315	49823.00
6471	47997.00
2118	47863.00
984	46780.00
7851	46661.00
5638	46280.00

Viewing the query plan

View the query plan for the previous query. Note the S3 Seq Scan, S3 HashAggregate, and S3 Query Scan steps that were run on the data on Amazon S3.

```
explain
select top 10 myspectrum_schema.sales.eventid, sum(myspectrum_schema.sales.pricepaid)
from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

```
QUERY PLAN
```

```

-----+
XN Limit  (cost=1001055770628.63..1001055770628.65 rows=10 width=31)

-> XN Merge  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

    Merge Key: sum(sales.derived_col2)

-> XN Network  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

    Send to leader

-> XN Sort  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

    Sort Key: sum(sales.derived_col2)

-> XN HashAggregate  (cost=1055770620.49..1055770620.99 rows=200
width=31)

    -> XN Hash Join DS_BCAST_INNER  (cost=3119.97..1055769620.49
rows=200000 width=31)

        Hash Cond: ("outer".derived_col1 = "inner".eventid)

-> XN S3 Query Scan sales  (cost=3010.00..5010.50
rows=200000 width=31)

    -> S3 HashAggregate  (cost=3010.00..3010.50
rows=200000 width=16)

        -> S3 Seq Scan spectrum.sales location:"s3://
redshift-downloads/ticket/spectrum/sales" format:TEXT  (cost=0.00..2150.00 rows=172000
width=16)

        Filter: (pricepaid > 30.00)

-> XN Hash  (cost=87.98..87.98 rows=8798 width=4)

    -> XN Seq Scan on event  (cost=0.00..87.98 rows=8798
width=4)

```

IAM policies for Amazon Redshift Spectrum

By default, Amazon Redshift Spectrum uses the AWS Glue Data Catalog in AWS Regions that support AWS Glue. In other AWS Regions, Redshift Spectrum uses the Athena Data Catalog. Your cluster needs authorization to access your external data catalog in AWS Glue or Athena and your data files in Amazon S3. You provide that authorization by referencing an AWS Identity and Access Management (IAM) role that is attached to your cluster. If you use an Apache Hive metastore to manage your data catalog, you don't need to provide access to Athena.

You can chain roles so that your cluster can assume other roles not attached to the cluster. For more information, see [Chaining IAM roles in Amazon Redshift Spectrum \(p. 263\)](#).

The AWS Glue catalog that you access might be encrypted to increase security. If the AWS Glue catalog is encrypted, you need the AWS KMS key for AWS Glue to access the AWS Glue Data Catalog. For more information, see [Encrypting Your AWS Glue Data Catalog](#) in the [AWS Glue Developer Guide](#).

Topics

- [Amazon S3 permissions \(p. 260\)](#)
- [Cross-account Amazon S3 permissions \(p. 260\)](#)
- [Policies to grant or restrict access using Redshift Spectrum \(p. 261\)](#)
- [Policies to grant minimum permissions \(p. 262\)](#)
- [Chaining IAM roles in Amazon Redshift Spectrum \(p. 263\)](#)
- [Controlling access to the AWS Glue Data Catalog \(p. 263\)](#)

Note

If you currently have Redshift Spectrum external tables in the Athena Data Catalog, you can migrate your Athena Data Catalog to an AWS Glue Data Catalog. To use the AWS Glue Data Catalog with Redshift Spectrum, you might need to change your IAM policies. For more information, see [Upgrading to the AWS Glue Data Catalog](#) in the [Athena User Guide](#).

Amazon S3 permissions

At a minimum, your cluster needs GET and LIST access to your Amazon S3 bucket. If your bucket is not in the same AWS account as your cluster, your bucket must also authorize your cluster to access the data. For more information, see [Authorizing Amazon Redshift to Access Other AWS Services on Your Behalf](#).

Note

The Amazon S3 bucket can't use a bucket policy that restricts access only from specific VPC endpoints.

The following policy grants GET and LIST access to any Amazon S3 bucket. The policy allows access to Amazon S3 buckets for Redshift Spectrum as well as COPY operations.

```
{  
  "Version": "2012-10-17",  
  "Statement": [{  
    "Effect": "Allow",  
    "Action": ["s3:Get*", "s3>List*"],  
    "Resource": "*"  
  }]  
}
```

The following policy grants GET and LIST access to your Amazon S3 bucket named myBucket.

```
{  
  "Version": "2012-10-17",  
  "Statement": [{  
    "Effect": "Allow",  
    "Action": ["s3:Get*", "s3>List*"],  
    "Resource": "arn:aws:s3:::myBucket/*"  
  }]  
}
```

Cross-account Amazon S3 permissions

To grant Redshift Spectrum permission to access data in an Amazon S3 bucket that belongs to another AWS account, add the following policy to the Amazon S3 bucket. For more information, see [Granting Cross-Account Bucket Permissions](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Example permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::redshift-account:role/spectrumrole"
            },
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3>ListMultipartUploadParts",
                "s3>ListBucket",
                "s3>ListBucketMultipartUploads"
            ],
            "Resource": [
                "arn:aws:s3:::bucketname",
                "arn:aws:s3:::bucketname/*"
            ]
        }
    ]
}
```

Policies to grant or restrict access using Redshift Spectrum

To grant access to an Amazon S3 bucket only using Redshift Spectrum, include a condition that allows access for the user agent AWS Redshift/Spectrum. The following policy allows access to Amazon S3 buckets only for Redshift Spectrum. It excludes other access, such as COPY operations.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["s3:Get*", "s3>List*"],
            "Resource": "arn:aws:s3:::myBucket/*",
            "Condition": {"StringEquals": {"aws:UserAgent": "AWS Redshift/Spectrum"}}
        }
    ]
}
```

Similarly, you might want to create an IAM role that allows access for COPY operations, but excludes Redshift Spectrum access. To do so, include a condition that denies access for the user agent **AWS Redshift/Spectrum**. The following policy allows access to an Amazon S3 bucket with the exception of Redshift Spectrum.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["s3:Get*", "s3>List*"],
            "Resource": "arn:aws:s3:::myBucket/*",
            "Condition": {"StringNotEquals": {"aws:UserAgent": "AWS Redshift/Spectrum"}}
        }
    ]
}
```

Policies to grant minimum permissions

The following policy grants the minimum permissions required to use Redshift Spectrum with Amazon S3, AWS Glue, and Athena.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3>ListMultipartUploadParts",
                "s3>ListBucket",
                "s3>ListBucketMultipartUploads"
            ],
            "Resource": [
                "arn:aws:s3:::bucketname",
                "arn:aws:s3:::bucketname/folder1/folder2/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue>CreateDatabase",
                "glue>DeleteDatabase",
                "glue>GetDatabase",
                "glue>GetDatabases",
                "glue>UpdateDatabase",
                "glue>CreateTable",
                "glue>DeleteTable",
                "glue>BatchDeleteTable",
                "glue>UpdateTable",
                "glue>GetTable",
                "glue>GetTables",
                "glue>BatchCreatePartition",
                "glue>CreatePartition",
                "glue>DeletePartition",
                "glue>BatchDeletePartition",
                "glue>UpdatePartition",
                "glue>GetPartition",
                "glue>GetPartitions",
                "glue>BatchGetPartition"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

If you use Athena for your data catalog instead of AWS Glue, the policy requires full Athena access. The following policy grants access to Athena resources. If your external database is in a Hive metastore, you don't need Athena access.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["athena:*"],
            "Resource": ["*"]
        }
    ]
}
```

```
}]  
}
```

Chaining IAM roles in Amazon Redshift Spectrum

When you attach a role to your cluster, your cluster can assume that role to access Amazon S3, Athena, and AWS Glue on your behalf. If a role attached to your cluster doesn't have access to the necessary resources, you can chain another role, possibly belonging to another account. Your cluster then temporarily assumes the chained role to access the data. You can also grant cross-account access by chaining roles. You can chain a maximum of 10 roles. Each role in the chain assumes the next role in the chain, until the cluster assumes the role at the end of chain.

To chain roles, you establish a trust relationship between the roles. A role that assumes another role must have a permissions policy that allows it to assume the specified role. In turn, the role that passes permissions must have a trust policy that allows it to pass its permissions to another role. For more information, see [Chaining IAM Roles in Amazon Redshift](#).

When you run the CREATE EXTERNAL SCHEMA command, you can chain roles by including a comma-separated list of role ARNs.

Note

The list of chained roles must not include spaces.

In the following example, MyRedshiftRole is attached to the cluster. MyRedshiftRole assumes the role AcmeData, which belongs to account 111122223333.

```
create external schema acme from data catalog
database 'acmedb' region 'us-west-2'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole,arn:aws:iam::111122223333:role/
AcmeData';
```

Controlling access to the AWS Glue Data Catalog

If you use AWS Glue for your data catalog, you can apply fine-grained access control to the AWS Glue Data Catalog with your IAM policy. For example, you might want to expose only a few databases and tables to a specific IAM role.

The following sections describe the IAM policies for various levels of access to data stored in the AWS Glue Data Catalog.

Topics

- [Policy for database operations \(p. 263\)](#)
- [Policy for table operations \(p. 264\)](#)
- [Policy for partition operations \(p. 267\)](#)

Policy for database operations

If you want to give users permissions to view and create a database, they need access rights to both the database and the AWS Glue Data Catalog.

The following example query creates a database.

```
CREATE EXTERNAL SCHEMA example_db
FROM DATA CATALOG DATABASE 'example_db' region 'us-west-2'
IAM_ROLE 'arn:aws:iam::redshift-account:role/spectrumrole'
CREATE EXTERNAL DATABASE IF NOT EXISTS
```

The following IAM policy gives the minimum permissions required for creating a database.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "glue:GetDatabase",  
                "glue>CreateDatabase"  
            ],  
            "Resource": [  
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",  
                "arn:aws:glue:us-west-2:redshift-account:catalog"  
            ]  
        }  
    ]  
}
```

The following example query lists the current databases.

```
SELECT * FROM SVV_EXTERNAL_DATABASES WHERE  
databasename = 'example_db1' or databasename = 'example_db2';
```

The following IAM policy gives the minimum permissions required to list the current databases.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "glue:GetDatabases"  
            ],  
            "Resource": [  
                "arn:aws:glue:us-west-2:redshift-account:database/example_db1",  
                "arn:aws:glue:us-west-2:redshift-account:database/example_db2",  
                "arn:aws:glue:us-west-2:redshift-account:catalog"  
            ]  
        }  
    ]  
}
```

Policy for table operations

If you want to give users permissions to view, create, drop, alter, or take other actions on tables, they need several types of access. They need access to the tables themselves, the databases they belong to, and the catalog.

The following example query creates an external table.

```
CREATE EXTERNAL TABLE example_db.example_tbl0(
    col0 INT,
    col1 VARCHAR(255)
) PARTITIONED BY (part INT) STORED AS TEXTFILE
LOCATION 's3://test/s3/location/';
```

The following IAM policy gives the minimum permissions required to create an external table.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue>CreateTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

The following example queries each list the current external tables.

```
SELECT * FROM svv_external_tables
WHERE tablename = 'example_tbl0' OR
tablename = 'example_tbl1';
```

```
SELECT * FROM svv_external_columns
WHERE tablename = 'example_tbl0' OR
tablename = 'example_tbl1';
```

```
SELECT parameters FROM svv_external_tables
WHERE tablename = 'example_tbl0' OR
tablename = 'example_tbl1';
```

The following IAM policy gives the minimum permissions required to list the current external tables.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [

```

```
        "arn:aws:glue:us-west-2:redshift-account:catalog",
        "arn:aws:glue:us-west-2:redshift-account:database/example_db",
        "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0",
        "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl1"

    ]
}
}
```

The following example query alters an existing table.

```
ALTER TABLE example_db.example_tbl0
SET TABLE PROPERTIES ('numRows' = '100');
```

The following IAM policy gives the minimum permissions required to alter an existing table.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetTable",
                "glue:UpdateTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

The following example query drops an existing table.

```
DROP TABLE example_db.example_tbl0;
```

The following IAM policy gives the minimum permissions required to drop an existing table.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:DeleteTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

```
        ]
    }
}
```

Policy for partition operations

If you want to give users permissions to perform partition-level operations (view, create, drop, alter, and so on), they need permissions to the tables that the partitions belong to. They also need permissions to the related databases and the AWS Glue Data Catalog.

The following example query creates a partition.

```
ALTER TABLE example_db.example_tbl0
ADD PARTITION (part=0) LOCATION 's3://test/s3/location/part=0/';
ALTER TABLE example_db.example_t
ADD PARTITION (part=1) LOCATION 's3://test/s3/location/part=1/';
```

The following IAM policy gives the minimum permissions required to create a partition.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetTable",
                "glue:BatchCreatePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

The following example query lists the current partitions.

```
SELECT * FROM svv_external_partitions
WHERE schemname = 'example_db' AND
tablename = 'example_tbl0'
```

The following IAM policy gives the minimum permissions required to list the current partitions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "
```

```

        "Action": [
            "glue:GetPartitions",
            "glue:GetTables",
            "glue:GetTable"
        ],
        "Resource": [
            "arn:aws:glue:us-west-2:redshift-account:catalog",
            "arn:aws:glue:us-west-2:redshift-account:database/example_db",
            "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
        ]
    }
}

```

The following example query alters an existing partition.

```
ALTER TABLE example_db.example_tbl0 PARTITION(part='0')
SET LOCATION 's3://test/s3/new/location/part=0/';
```

The following IAM policy gives the minimum permissions required to alter an existing partition.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetPartition",
                "glue:UpdatePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

The following example query drops an existing partition.

```
ALTER TABLE example_db.example_tbl0 DROP PARTITION(part='0');
```

The following IAM policy gives the minimum permissions required to drop an existing partition.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue>DeletePartition"
            ]
        }
    ]
}
```

```
        ],
        "Resource": [
            "arn:aws:glue:us-west-2:redshift-account:catalog",
            "arn:aws:glue:us-west-2:redshift-account:database/example_db",
            "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
        ]
    }
}
```

Using Redshift Spectrum with AWS Lake Formation

You can use AWS Lake Formation to centrally define and enforce database, table, and column-level access policies to data stored in Amazon S3. After your data is registered with an AWS Glue Data Catalog enabled with Lake Formation, you can query it by using several services, including Redshift Spectrum.

Lake Formation provides the security and governance of the Data Catalog. Within Lake Formation, you can grant and revoke permissions to the Data Catalog objects, such as databases, tables, columns, and underlying Amazon S3 storage.

Important

You can only use Redshift Spectrum with a Lake Formation enabled Data Catalog in AWS Regions where Lake Formation is available. For a list of available Regions, see [AWS Lake Formation endpoints and quotas](#) in the *AWS General Reference*.

By using Redshift Spectrum with Lake Formation, you can do the following:

- Use Lake Formation as a centralized place where you grant and revoke permissions and access control policies on all of your data in the data lake. Lake Formation provides a hierarchy of permissions to control access to databases and tables in a Data Catalog. For more information, see [Overview of Lake Formation permissions](#) in the *AWS Lake Formation Developer Guide*.
- Create external tables and run queries on data in the data lake. Before users in your account can run queries, a data lake account administrator registers your existing Amazon S3 paths containing source data with Lake Formation. The administrator also creates tables and grants permissions to your users. Access can be granted on databases, tables, or columns. The administrator can use data filters in Lake Formation to grant granular access control over your sensitive data stored in Amazon S3. For more information, see [Using data filters for row-level and cell-level security \(p. 270\)](#).

After the data is registered in the Data Catalog, each time users try to run queries, Lake Formation verifies access to the table for that specific principal. Lake Formation vends temporary credentials to Redshift Spectrum, and the query runs.

When you use Redshift Spectrum with a Data Catalog enabled for Lake Formation, one of the following must be in place:

- An IAM role associated with the cluster that has permission to the Data Catalog.
- A federated IAM identity configured to manage access to external resources. For more information, see [Using a federated identity to manage Amazon Redshift access to local resources and Amazon Redshift external tables](#).

Important

You can't chain IAM roles when using Redshift Spectrum with a Data Catalog enabled for Lake Formation.

To learn more about the steps required to set up AWS Lake Formation to use with Redshift Spectrum, see [Tutorial: Creating a data lake from a JDBC Source in Lake Formation](#) in the *AWS Lake Formation Developer Guide*. Specifically, see [Query the data in the data lake using Amazon Redshift Spectrum](#) for details about integration with Redshift Spectrum. The data and AWS resources used in this topic depend on previous steps in the tutorial.

Using data filters for row-level and cell-level security

You can define data filters in AWS Lake Formation to control your Redshift Spectrum queries' row-level and cell-level access to data defined in your Data Catalog. To set this up, you perform the following tasks:

- Create a data filter in Lake Formation with the following information:
 - A column specification with a list of columns to include or exclude from query results.
 - A row filter expression that specifies the rows to include in the query results.

For more information about how to create a data filter, see [Data filters in Lake Formation](#) in the *AWS Lake Formation Developer Guide*.

- Create an external table in Amazon Redshift that references a table in your Lake Formation enabled Data Catalog. For details on how to query a Lake Formation table using Redshift Spectrum, see [Query the data in the data lake using Amazon Redshift Spectrum](#) in the *AWS Lake Formation Developer Guide*.

After the table is defined in Amazon Redshift, you can query the Lake Formation table and access only the rows and columns that are allowed by the data filter.

For a detailed guide on how to set up row-level and cell-level security in Lake Formation, and then query using Redshift Spectrum, see [Use Amazon Redshift Spectrum with row-level and cell-level security policies defined in AWS Lake Formation](#).

Creating data files for queries in Amazon Redshift Spectrum

The data files that you use for queries in Amazon Redshift Spectrum are commonly the same types of files that you use for other applications. For example, the same types of files are used with Amazon Athena, Amazon EMR, and Amazon QuickSight. You can query the data in its original format directly from Amazon S3. To do this, the data files must be in a format that Redshift Spectrum supports and be located in an Amazon S3 bucket that your cluster can access.

The Amazon S3 bucket with the data files and the Amazon Redshift cluster must be in the same AWS Region. For information about supported AWS Regions, see [Amazon Redshift Spectrum Regions \(p. 249\)](#).

Data formats for Redshift Spectrum

Redshift Spectrum supports the following structured and semistructured data formats.

File format	Columnar	Supports parallel reads	Split unit
Parquet	Yes	Yes	Row group
ORC	Yes	Yes	Stripe

File format	Columnar	Supports parallel reads	Split unit
RCFile	Yes	Yes	Row group
TextFile	No	Yes	Row
SequenceFile	No	Yes	Row or block
RegexSerde	No	Yes	Row
OpenCSV	No	Yes	Row
AVRO	No	Yes	Block
Ion	No	No	N/A
JSON	No	No	N/A

In the preceding table, the headings indicate the following:

- **Columnar** – Whether the file format physically stores data in a column-oriented structure as opposed to a row-oriented one.
- **Supports parallel reads** – Whether the file format supports reading individual blocks within the file. Reading individual blocks enables the distributed processing of a file across multiple independent Redshift Spectrum requests instead of having to read the full file in a single request.
- **Split unit** – For file formats that can be read in parallel, the split unit is the smallest chunk of data that a single Redshift Spectrum request can process.

Note

Timestamp values in text files must be in the format yyyy-MM-dd HH:mm:ss.SSSSSS, as the following timestamp value shows: 2017-05-01 11:30:59.000000.

We recommend using a columnar storage file format, such as Apache Parquet. With a columnar storage file format, you can minimize data transfer out of Amazon S3 by selecting only the columns that you need.

Compression types for Redshift Spectrum

To reduce storage space, improve performance, and minimize costs, we strongly recommend that you compress your data files. Redshift Spectrum recognizes file compression types based on the file extension.

Redshift Spectrum supports the following compression types and extensions.

Compression Algorithm	File Extension	Supports Parallel Reads
Gzip	.gz	No
Bzip2	.bz2	Yes
Snappy	.snappy	No

You can apply compression at different levels. Most commonly, you compress a whole file or compress individual blocks within a file. Compressing columnar formats at the file level doesn't yield performance benefits.

For Redshift Spectrum to be able to read a file in parallel, the following must be true:

- The file format supports parallel reads.
- The file-level compression, if any, supports parallel reads.

It doesn't matter whether the individual split units within a file are compressed using a compression algorithm that can be read in parallel, because each split unit is processed by a single Redshift Spectrum request. An example of this is Snappy-compressed Parquet files. Individual row groups within the Parquet file are compressed using Snappy, but the top-level structure of the file remains uncompressed. In this case, the file can be read in parallel because each Redshift Spectrum request can read and process individual row groups from Amazon S3.

Encryption for Redshift Spectrum

Redshift Spectrum transparently decrypts data files that are encrypted using the following encryption options:

- Server-side encryption (SSE-S3) using an AES-256 encryption key managed by Amazon S3.
- Server-side encryption with keys managed by AWS Key Management Service (SSE-KMS).

Redshift Spectrum doesn't support Amazon S3 client-side encryption. For more information on server-side encryption, see [Protecting Data Using Server-Side Encryption](#) in the *Amazon Simple Storage Service User Guide*.

Amazon Redshift uses massively parallel processing (MPP) to achieve fast execution of complex queries operating on large amounts of data. Redshift Spectrum extends the same principle to query external data, using multiple Redshift Spectrum instances as needed to scan files. Place the files in a separate folder for each table.

You can optimize your data for parallel processing by doing the following:

- If your file format or compression doesn't support reading in parallel, break large files into many smaller files. We recommend using file sizes between 64 MB and 1 GB.
- Keep all the files about the same size. If some files are much larger than others, Redshift Spectrum can't distribute the workload evenly.

Creating external schemas for Amazon Redshift Spectrum

All external tables must be created in an external schema, which you create using a [CREATE EXTERNAL SCHEMA \(p. 762\)](#) statement.

Note

Some applications use the term *database* and *schema* interchangeably. In Amazon Redshift, we use the term *schema*.

An Amazon Redshift external schema references an external database in an external data catalog. You can create the external database in Amazon Redshift, in [Amazon Athena](#), in [AWS Glue Data Catalog](#), or in an Apache Hive metastore, such as [Amazon EMR](#). If you create an external database in Amazon Redshift, the database resides in the Athena Data Catalog. To create a database in a Hive metastore, you need to create the database in your Hive application.

Amazon Redshift needs authorization to access the Data Catalog in Athena and the data files in Amazon S3 on your behalf. To provide that authorization, you first create an AWS Identity and Access

Management (IAM) role. Then you attach the role to your cluster and provide Amazon Resource Name (ARN) for the role in the Amazon Redshift CREATE EXTERNAL SCHEMA statement. For more information about authorization, see [IAM policies for Amazon Redshift Spectrum \(p. 259\)](#).

Note

If you currently have Redshift Spectrum external tables in the Athena Data Catalog, you can migrate your Athena Data Catalog to an AWS Glue Data Catalog. To use an AWS Glue Data Catalog with Redshift Spectrum, you might need to change your IAM policies. For more information, see [Upgrading to the AWS Glue Data Catalog in the Amazon Athena User Guide](#).

To create an external database at the same time you create an external schema, specify FROM DATA CATALOG and include the CREATE EXTERNAL DATABASE clause in your CREATE EXTERNAL SCHEMA statement.

The following example creates an external schema named spectrum_schema using the external database spectrum_db.

```
create external schema spectrum_schema from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
create external database if not exists;
```

If you manage your data catalog using Athena, specify the Athena database name and the AWS Region in which the Athena Data Catalog is located.

The following example creates an external schema using the default sampledb database in the Athena Data Catalog.

```
create external schema athena_schema from data catalog
database 'sampledb'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
region 'us-east-2';
```

Note

The region parameter references the AWS Region in which the Athena Data Catalog is located, not the location of the data files in Amazon S3.

If you manage your data catalog using a Hive metastore, such as Amazon EMR, your security groups must be configured to allow traffic between the clusters.

In the CREATE EXTERNAL SCHEMA statement, specify FROM HIVE METASTORE and include the metastore's URI and port number. The following example creates an external schema using a Hive metastore database named hive_db.

```
create external schema hive_schema
from hive metastore
database 'hive_db'
uri '172.10.10.10' port 99
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
```

To view external schemas for your cluster, query the PG_EXTERNAL_SCHEMA catalog table or the SVV_EXTERNAL_SCHEMAS view. The following example queries SVV_EXTERNAL_SCHEMAS, which joins PG_EXTERNAL_SCHEMA and PG_NAMESPACE.

```
select * from svv_external_schemas
```

For the full command syntax and examples, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#).

Working with external catalogs in Amazon Redshift Spectrum

The metadata for Amazon Redshift Spectrum external databases and external tables is stored in an external data catalog. By default, Redshift Spectrum metadata is stored in an Athena Data Catalog. You can view and manage Redshift Spectrum databases and tables in your Athena console.

You can also create and manage external databases and external tables using Hive data definition language (DDL) using Athena or a Hive metastore, such as Amazon EMR.

Note

We recommend using Amazon Redshift to create and manage external databases and external tables in Redshift Spectrum.

Viewing Redshift Spectrum databases in Athena and AWS Glue

You can create an external database by including the CREATE EXTERNAL DATABASE IF NOT EXISTS clause as part of your CREATE EXTERNAL SCHEMA statement. In such cases, the external database metadata is stored in your Data Catalog. The metadata for external tables that you create qualified by the external schema is also stored in your Data Catalog.

Athena and AWS Glue maintain a Data Catalog for each supported AWS Region. To view table metadata, log on to the Athena or AWS Glue console. In Athena, choose **Data sources**, your AWS Glue, then view the details of your database. In AWS Glue, choose **Databases**, your external database, then view the details of your database.

If you create and manage your external tables using Athena, register the database using CREATE EXTERNAL SCHEMA. For example, the following command registers the Athena database named `sampled`.

```
create external schema athena_sample
from data catalog
database 'sampledb'
iam_role 'arn:aws:iam::123456789012:role/mySpectrumRole'
region 'us-east-1';
```

When you query the `SVV_EXTERNAL_TABLES` system view, you see tables in the Athena `sampledb` database and also tables that you created in Amazon Redshift.

```
select * from svv_external_tables;
```

schemaName	tableName	location
athena_sample	elb_logs	s3://athena-examples/elb/plain/text
athena_sample	lineitem_1t_csv	s3://myspectrum/tpch/1000/lineitem_csv
athena_sample	lineitem_1t_part	s3://myspectrum/tpch/1000/lineitem_partition
spectrum	sales	s3://redshift-downloads/ticket/spectrum/sales
spectrum	sales_part	s3://redshift-downloads/ticket/spectrum/sales_part

Registering an Apache Hive metastore database

If you create external tables in an Apache Hive metastore, you can use CREATE EXTERNAL SCHEMA to register those tables in Redshift Spectrum.

In the CREATE EXTERNAL SCHEMA statement, specify the FROM HIVE METASTORE clause and provide the Hive metastore URI and port number. The IAM role must include permission to access Amazon S3 but doesn't need any Athena permissions. The following example registers a Hive metastore.

```
create external schema if not exists hive_schema
from hive metastore
database 'hive_database'
uri 'ip-10-0-111-111.us-west-2.compute.internal' port 9083
iam_role 'arn:aws:iam::123456789012:role/mySpectrumRole';
```

Enabling your Amazon Redshift cluster to access your Amazon EMR cluster

If your Hive metastore is in Amazon EMR, you must give your Amazon Redshift cluster access to your Amazon EMR cluster. To do so, you create an Amazon EC2 security group. You then allow all inbound traffic to the EC2 security group from your Amazon Redshift cluster's security group and your Amazon EMR cluster's security group. Then you add the EC2 security to both your Amazon Redshift cluster and your Amazon EMR cluster.

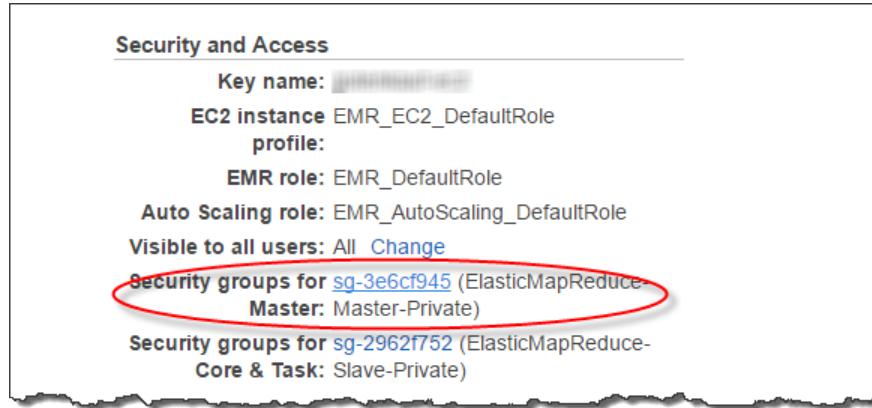
View your Amazon Redshift cluster's security group name

To display the security group, do the following:

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Clusters**, then choose the cluster from the list to open its details.
3. Choose **Properties** and view the **Network and security settings** section.
4. Find your security group in **VPC security group** and take note of it.

View the Amazon EMR master node security group name

1. Open your Amazon EMR cluster. For more information, see [Use security configurations to set up cluster security](#) in the *Amazon EMR Management Guide*.
2. Under **Security and access**, make a note of the Amazon EMR master node security group name.



To create or modify an Amazon EC2 security group to allow connection between Amazon Redshift and Amazon EMR

1. In the Amazon EC2 dashboard, choose **Security groups**. For more information, see [Security group rules](#) in the *Amazon EC2 User Guide for Linux Instances*
2. Choose **Create security group**.

3. If you are using VPC, choose the VPC that your Amazon Redshift and Amazon EMR clusters are in.
 4. Add an inbound rule.
 1. For **Type**, choose **Custom TCP**.
 2. For **Source**, choose **Custom**.
 3. Enter the name of your Amazon Redshift security group.
 5. Add another inbound rule.
 1. For **Type**, choose **TCP**.
 2. For **Port Range**, enter **9083**.
- Note**
The default port for an EMR HMS is 9083. If your HMS uses a different port, specify that port in the inbound rule and in the external schema definition.
3. For **Source**, choose **Custom**.
 6. Enter a security group name and description.
 7. Choose **Create security group**.

To add the Amazon EC2 security group you created in the previous procedure to your Amazon Redshift cluster

1. In Amazon Redshift, choose your cluster.
2. Choose **Properties**.
3. View the **Network and security settings** and choose **Edit**.
4. In **VPC security group**, choose the new security group name.
5. Choose **Save changes**.

To add the Amazon EC2 security group to your Amazon EMR cluster

1. In Amazon EMR, choose your cluster. For more information, see [Use security configurations to set up cluster security](#) in the *Amazon EMR Management Guide*.
2. Under **Hardware**, choose the link for the Master node.
3. Choose the link in the **EC2 instance ID** column.

The screenshot shows the 'Hardware' section of the Amazon EMR console. It displays a table of instances. The first column is labeled 'ID' and the second column is labeled 'EC2 Instance Id'. The 'EC2 Instance Id' for the first row is 'i-0fdf5aaae0eafdf03', which is circled in red. Other columns include 'EBS Volumes per Instance', 'Status', and 'Public Dn'.

ID	EC2 Instance Id	EBS Volumes per Instance	Status	Public Dn
ci-1ZU2SYYO6Y9D	i-0fdf5aaae0eafdf03	0	View All	Running

4. For **Actions**, choose **Security, Change security groups**.
5. In **Associated security groups**, choose the new security group, and choose **Add security group**.
6. Choose **Save**.

Creating external tables for Redshift Spectrum

You create an external table in an external schema. To create external tables, you must be the owner of the external schema or a superuser. To transfer ownership of an external schema, use [ALTER SCHEMA \(p. 638\)](#) to change the owner. The following example changes the owner of the spectrum_schema schema to newowner.

```
alter schema spectrum_schema owner to newowner;
```

To run a Redshift Spectrum query, you need the following permissions:

- Usage permission on the schema
- Permission to create temporary tables in the current database

The following example grants usage permission on the schema spectrum_schema to the spectrumusers user group.

```
grant usage on schema spectrum_schema to group spectrumusers;
```

The following example grants temporary permission on the database spectrumdb to the spectrumusers user group.

```
grant temp on database spectrumdb to group spectrumusers;
```

You can create an external table in Amazon Redshift, AWS Glue, Amazon Athena, or an Apache Hive metastore. For more information, see [Getting Started Using AWS Glue](#) in the *AWS Glue Developer Guide*, [Getting Started](#) in the *Amazon Athena User Guide*, or [Apache Hive](#) in the *Amazon EMR Developer Guide*.

If your external table is defined in AWS Glue, Athena, or a Hive metastore, you first create an external schema that references the external database. Then you can reference the external table in your SELECT statement by prefixing the table name with the schema name, without needing to create the table in Amazon Redshift. For more information, see [Creating external schemas for Amazon Redshift Spectrum \(p. 272\)](#).

To allow Amazon Redshift to view tables in the AWS Glue Data Catalog, add glue:GetTable to the Amazon Redshift IAM role. Otherwise you might get an error similar to the following.

```
RedshiftIamRoleSession is not authorized to perform: glue:GetTable on resource: *;
```

For example, suppose that you have an external table named lineitem_athena defined in an Athena external catalog. In this case, you can define an external schema named athena_schema, then query the table using the following SELECT statement.

```
select count(*) from athena_schema.lineitem_athena;
```

To define an external table in Amazon Redshift, use the [CREATE EXTERNAL TABLE \(p. 768\)](#) command. The external table statement defines the table columns, the format of your data files, and the location of your data in Amazon S3. Redshift Spectrum scans the files in the specified folder and any subfolders. Redshift Spectrum ignores hidden files and files that begin with a period, underscore, or hash mark (. , _ or #) or end with a tilde (~).

The following example creates a table named SALES in the Amazon Redshift external schema named spectrum. The data is in tab-delimited text files.

```
create external table spectrum.sales(
    salesid integer,
    listid integer,
    sellerid integer,
    buyerid integer,
    eventid integer,
    dateid smallint,
    qtypsold smallint,
    pricepaid decimal(8,2),
    commission decimal(8,2),
    saletime timestamp)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://redshift-downloads/ticket/spectrum/sales/'
table properties ('numRows'='172000');
```

To view external tables, query the [SVV_EXTERNAL_TABLES \(p. 1431\)](#) system view.

Pseudocolumns

By default, Amazon Redshift creates external tables with the pseudocolumns \$path, \$size, and \$spectrum_oid. Select the \$path column to view the path to the data files on Amazon S3, and select the \$size column to view the size of the data files for each row returned by a query. The \$spectrum_oid column provides the ability to perform correlated queries with Redshift Spectrum. For an example, see [Example: Performing correlated subqueries in Redshift Spectrum \(p. 290\)](#). You must delimit the \$path, \$size, and \$spectrum_oid column names with double quotation marks. A SELECT * clause doesn't return the pseudocolumns. You must explicitly include the \$path, \$size, and \$spectrum_oid column names in your query, as the following example shows.

```
select "$path", "$size", "$spectrum_oid"
from spectrum.sales_part where saledate = '2008-12-01';
```

You can disable the creation of pseudocolumns for a session by setting the spectrum_enable_pseudo_columns configuration parameter to false. For more information, see [spectrum_enable_pseudo_columns \(p. 1697\)](#). You can also disable only the \$spectrum_oid pseudocolumn by setting the enable_spectrum_oid to false. For more information, see [enable_spectrum_oid \(p. 1698\)](#). However, disabling the \$spectrum_oid pseudocolumn also disables support for correlated queries with Redshift Spectrum.

Important

Selecting \$size, \$path, or \$spectrum_oid incurs charges because Redshift Spectrum scans the data files on Amazon S3 to determine the size of the result set. For more information, see [Amazon Redshift Pricing](#).

Pseudocolumns example

The following example returns the total size of related data files for an external table.

```
select distinct "$path", "$size"
from spectrum.sales_part;

$path                                         | $size
-----+-----
s3://redshift-downloads/ticket/spectrum/sales_partition/saledate=2008-01/ | 1616
s3://redshift-downloads/ticket/spectrum/sales_partition/saledate=2008-02/ | 1444
s3://redshift-downloads/ticket/spectrum/sales_partition/saledate=2008-03/ | 1644
```

Partitioning Redshift Spectrum external tables

When you partition your data, you can restrict the amount of data that Redshift Spectrum scans by filtering on the partition key. You can partition your data by any key.

A common practice is to partition the data based on time. For example, you might choose to partition by year, month, date, and hour. If you have data coming from multiple sources, you might partition by a data source identifier and date.

The following procedure describes how to partition your data.

To partition your data

1. Store your data in folders in Amazon S3 according to your partition key.

Create one folder for each partition value and name the folder with the partition key and value. For example, if you partition by date, you might have folders named `saledate=2017-04-01`, `saledate=2017-04-02`, and so on. Redshift Spectrum scans the files in the partition folder and any subfolders. Redshift Spectrum ignores hidden files and files that begin with a period, underscore, or hash mark (. , _ or #) or end with a tilde (~).

2. Create an external table and specify the partition key in the PARTITIONED BY clause.

The partition key can't be the name of a table column. The data type can be SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, or TIMESTAMP data type.

3. Add the partitions.

Using [ALTER TABLE \(p. 640\)](#) ... ADD PARTITION, add each partition, specifying the partition column and key value, and the location of the partition folder in Amazon S3. You can add multiple partitions in a single ALTER TABLE ... ADD statement. The following example adds partitions for '2008-01' and '2008-03'.

```
alter table spectrum.sales_part add
partition(saledate='2008-01-01')
location 's3://redshift-downloads/ticket/spectrum/sales_partition/saledate=2008-01/'
partition(saledate='2008-03-01')
location 's3://redshift-downloads/ticket/spectrum/sales_partition/saledate=2008-03/';
```

Note

If you use the AWS Glue catalog, you can add up to 100 partitions using a single ALTER TABLE statement.

Partitioning data examples

In this example, you create an external table that is partitioned by a single partition key and an external table that is partitioned by two partition keys.

The sample data for this example is located in an Amazon S3 bucket that gives read access to all authenticated AWS users. Your cluster and your external data files must be in the same AWS Region. The sample data bucket is in the US East (N. Virginia) Region (us-east-1). To access the data using Redshift Spectrum, your cluster must also be in us-east-1. To list the folders in Amazon S3, run the following command.

```
aws s3 ls s3://redshift-downloads/ticket/spectrum/sales_partition/
```

```
PRE saledate=2008-01/
```

```
PRE saledate=2008-03/
PRE saledate=2008-04/
PRE saledate=2008-05/
PRE saledate=2008-06/
PRE saledate=2008-12/
```

If you don't already have an external schema, run the following command. Substitute the Amazon Resource Name (ARN) for your AWS Identity and Access Management (IAM) role.

```
create external schema spectrum
from data catalog
database 'spectrumbdb'
iam_role 'arn:aws:iam::123456789012:role/myspectrumrole'
create external database if not exists;
```

Example 1: Partitioning with a single partition key

In the following example, you create an external table that is partitioned by month.

To create an external table partitioned by month, run the following command.

```
create external table spectrum.sales_part(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
partitioned by (saledate char(10))
row format delimited
fields terminated by '|'
stored as textfile
location 's3://redshift-downloads/ticket/spectrum/sales_partition/'
table properties ('numRows'='172000');
```

To add the partitions, run the following ALTER TABLE command.

```
alter table spectrum.sales_part add
partition(saledate='2008-01')
location 's3://redshift-downloads/ticket/spectrum/sales_partition/saledate=2008-01/'

partition(saledate='2008-03')
location 's3://redshift-downloads/ticket/spectrum/sales_partition/saledate=2008-03/'

partition(saledate='2008-04')
location 's3://redshift-downloads/ticket/spectrum/sales_partition/saledate=2008-04/';
```

To select data from the partitioned table, run the following query.

```
select top 5 spectrum.sales_part.eventid, sum(spectrum.sales_part.pricepaid)
from spectrum.sales_part, event
where spectrum.sales_part.eventid = event.eventid
and spectrum.sales_part.pricepaid > 30
and saledate = '2008-01'
group by spectrum.sales_part.eventid
order by 2 desc;
```

```
eventid | sum
-----+-----
 4124 | 21179.00
 1924 | 20569.00
 2294 | 18830.00
 2260 | 17669.00
 6032 | 17265.00
```

To view external table partitions, query the [SVV_EXTERNAL_PARTITIONS \(p. 1429\)](#) system view.

```
select schemaname, tablename, values, location from svv_external_partitions
where tablename = 'sales_part';
```

schemaname	tablename	values	location
spectrum	sales_part	[{"2008-01"}]	s3://redshift-downloads/ticket/spectrum/sales_partition/saledate=2008-01
spectrum	sales_part	[{"2008-03"}]	s3://redshift-downloads/ticket/spectrum/sales_partition/saledate=2008-03
spectrum	sales_part	[{"2008-04"}]	s3://redshift-downloads/ticket/spectrum/sales_partition/saledate=2008-04

Example 2: Partitioning with a multiple partition key

To create an external table partitioned by date and eventid, run the following command.

```
create external table spectrum.sales_event(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
partitioned by (salesmonth char(10), event integer)
row format delimited
fields terminated by '|'
stored as textfile
location 's3://redshift-downloads/ticket/spectrum/salesevent/'
table properties ('numRows'='172000');
```

To add the partitions, run the following ALTER TABLE command.

```
alter table spectrum.sales_event add
partition(salesmonth='2008-01', event='101')
location 's3://redshift-downloads/ticket/spectrum/salesevent/salesmonth=2008-01/event=101/'

partition(salesmonth='2008-01', event='102')
location 's3://redshift-downloads/ticket/spectrum/salesevent/salesmonth=2008-01/event=102/'

partition(salesmonth='2008-01', event='103')
location 's3://redshift-downloads/ticket/spectrum/salesevent/salesmonth=2008-01/event=103/'

partition(salesmonth='2008-02', event='101')
location 's3://redshift-downloads/ticket/spectrum/salesevent/salesmonth=2008-02/event=101/'
```

```

partition(salesmonth='2008-02', event='102')
location 's3://redshift-downloads/ticket/spectrum/salesevent/salesmonth=2008-02/event=102/'

partition(salesmonth='2008-02', event='103')
location 's3://redshift-downloads/ticket/spectrum/salesevent/salesmonth=2008-02/event=103/'

partition(salesmonth='2008-03', event='101')
location 's3://redshift-downloads/ticket/spectrum/salesevent/salesmonth=2008-03/event=101/'

partition(salesmonth='2008-03', event='102')
location 's3://redshift-downloads/ticket/spectrum/salesevent/salesmonth=2008-03/event=102/'

partition(salesmonth='2008-03', event='103')
location 's3://redshift-downloads/ticket/spectrum/salesevent/salesmonth=2008-03/
event=103';

```

Run the following query to select data from the partitioned table.

```

select spectrum.sales_event.salesmonth, event.eventname,
sum(spectrum.sales_event.pricepaid)
from spectrum.sales_event, event
where spectrum.sales_event.eventid = event.eventid
and salesmonth = '2008-02'
and (event = '101'
or event = '102'
or event = '103')
group by event.eventname, spectrum.sales_event.salesmonth
order by 3 desc;

```

salesmonth	eventname	sum
2008-02	The Magic Flute	5062.00
2008-02	La Sonnambula	3498.00
2008-02	Die Walkure	534.00

Mapping external table columns to ORC columns

You use Amazon Redshift Spectrum external tables to query data from files in ORC format. Optimized row columnar (ORC) format is a columnar storage file format that supports nested data structures. For more information about querying nested data, see [Querying Nested Data with Amazon Redshift Spectrum \(p. 294\)](#).

When you create an external table that references data in an ORC file, you map each column in the external table to a column in the ORC data. To do so, you use one of the following methods:

- [Mapping by position \(p. 282\)](#)
- [Mapping by column name \(p. 284\)](#)

Mapping by column name is the default.

Mapping by position

With position mapping, the first column defined in the external table maps to the first column in the ORC data file, the second to the second, and so on. Mapping by position requires that the order of columns in the external table and in the ORC file match. If the order of the columns doesn't match, then you can map the columns by name.

Important

In earlier releases, Redshift Spectrum used position mapping by default. If you need to continue using position mapping for existing tables, set the table property `orc.schema.resolution` to `position`, as the following example shows.

```
alter table spectrum.orc_example
set table properties('orc.schema.resolution'='position');
```

For example, the table `SPECTRUM.ORC_EXAMPLE` is defined as follows.

```
create external table spectrum.orc_example(
int_col int,
float_col float,
nested_col struct<
    "int_col" : int,
    "map_col" : map<int, array<float >>
    >
) stored as orc
location 's3://example/orc/files/';
```

The table structure can be abstracted as follows.

- 'int_col' : int
- 'float_col' : float
- 'nested_col' : struct
 - 'int_col' : int
 - 'map_col' : map
 - key : int
 - value : array
 - value : float

The underlying ORC file has the following file structure.

- ORC file root(id = 0)
 - 'int_col' : int (id = 1)
 - 'float_col' : float (id = 2)
 - 'nested_col' : struct (id = 3)
 - 'int_col' : int (id = 4)
 - 'map_col' : map (id = 5)
 - key : int (id = 6)
 - value : array (id = 7)
 - value : float (id = 8)

In this example, you can map each column in the external table to a column in ORC file strictly by position. The following shows the mapping.

External table column name	ORC column ID	ORC column name
int_col	1	int_col
float_col	2	float_col
nested_col	3	nested_col
nested_col.int_col	4	int_col
nested_col.map_col	5	map_col
nested_col.map_col.key	6	NA

External table column name	ORC column ID	ORC column name
nested_col.map_col.value	7	NA
nested_col.map_col.value.item	8	NA

Mapping by column name

Using name mapping, you map columns in an external table to named columns in ORC files on the same level, with the same name.

For example, suppose that you want to map the table from the previous example, SPECTRUM.ORC_EXAMPLE, with an ORC file that uses the following file structure.

- ```

• ORC file root(id = 0)
 o 'nested_col' : struct (id = 1)
 - 'map_col' : map (id = 2)
 - key : int (id = 3)
 - value : array (id = 4)
 - value : float (id = 5)
 o 'int_col' : int (id = 6)
 o 'int_col' : int (id = 7)
 o 'float_col' : float (id = 8)

```

Using position mapping, Redshift Spectrum attempts the following mapping.

| External table column name | ORC column ID | ORC column name |
|----------------------------|---------------|-----------------|
| int_col                    | 1             | struct          |
| float_col                  | 7             | int_col         |
| nested_col                 | 8             | float_col       |

When you query a table with the preceding position mapping, the SELECT command fails on type validation because the structures are different.

You can map the same external table to both file structures shown in the previous examples by using column name mapping. The table columns int\_col, float\_col, and nested\_col map by column name to columns with the same names in the ORC file. The column named nested\_col in the external table is a struct column with subcolumns named map\_col and int\_col. The subcolumns also map correctly to the corresponding columns in the ORC file by column name.

## Creating external tables for data managed in Apache Hudi

To query data in Apache Hudi Copy On Write (CoW) format, you can use Amazon Redshift Spectrum external tables. A Hudi Copy On Write table is a collection of Apache Parquet files stored in Amazon S3. You can read Copy On Write (CoW) tables in Apache Hudi versions 0.5.2, 0.6.0, 0.7.0, 0.8.0, 0.9.0, 0.10.0, 0.10.1, 0.11.0, and 0.11.1 that are created and modified with insert, delete, and upsert write operations. For example, bootstrap tables are not supported. For more information, see [Copy On Write Table](#) in the open source Apache Hudi documentation.

When you create an external table that references data in Hudi CoW format, you map each column in the external table to a column in the Hudi data. Mapping is done by column.

The data definition language (DDL) statements for partitioned and unpartitioned Hudi tables are similar to those for other Apache Parquet file formats. For Hudi tables, you define INPUTFORMAT as `org.apache.hudi.hadoop.HoodieParquetInputFormat`. The LOCATION parameter must point to the Hudi table base folder that contains the `.hoodie` folder, which is required to establish the Hudi commit timeline. In some cases, a SELECT operation on a Hudi table might fail with the message No valid Hudi commit timeline found. If so, check if the `.hoodie` folder is in the correct location and contains a valid Hudi commit timeline.

**Note**

Apache Hudi format is only supported when you use an AWS Glue Data Catalog. It's not supported when you use an Apache Hive metastore as the external catalog.

The DDL to define an unpartitioned table has the following format.

```
CREATE EXTERNAL TABLE tbl_name (columns)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://s3-bucket/prefix'
```

The DDL to define a partitioned table has the following format.

```
CREATE EXTERNAL TABLE tbl_name (columns)
PARTITIONED BY(pcolumn1 pcolumn1-type[,...])
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://s3-bucket/prefix'
```

To add partitions to a partitioned Hudi table, run an ALTER TABLE ADD PARTITION command where the LOCATION parameter points to the Amazon S3 subfolder with the files that belong to the partition.

The DDL to add partitions has the following format.

```
ALTER TABLE tbl_name
ADD IF NOT EXISTS PARTITION(pcolumn1=pvalue1[,...])
LOCATION 's3://s3-bucket/prefix/partition-path'
```

## Creating external tables for data managed in Delta Lake

To query data in Delta Lake tables, you can use Amazon Redshift Spectrum external tables.

To access a Delta Lake table from Redshift Spectrum, generate a manifest before the query. A Delta Lake *manifest* contains a listing of files that make up a consistent snapshot of the Delta Lake table. In a partitioned table, there is one manifest per partition. A Delta Lake table is a collection of Apache Parquet files stored in Amazon S3. For more information, see [Delta Lake](#) in the open source Delta Lake documentation.

When you create an external table that references data in Delta Lake tables, you map each column in the external table to a column in the Delta Lake table. Mapping is done by column name.

The DDL for partitioned and unpartitioned Delta Lake tables is similar to that for other Apache Parquet file formats. For Delta Lake tables, you define INPUTFORMAT as `org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat` and OUTPUTFORMAT as `org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat`. The LOCATION parameter

must point to the manifest folder in the table base folder. If a SELECT operation on a Delta Lake table fails, for possible reasons see [Limitations and troubleshooting for Delta Lake tables \(p. 286\)](#).

The DDL to define an unpartitioned table has the following format.

```
CREATE EXTERNAL TABLE tbl_name (columns)
ROW FORMAT SERDE 'org.apache.hadoop.hive ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hadoop.hive ql.io.SymlinkTextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://s3-bucket/prefix/_symlink_format_manifest'
```

The DDL to define a partitioned table has the following format.

```
CREATE EXTERNAL TABLE tbl_name (columns)
PARTITIONED BY(pcolumn1 pcolumn1-type[,...])
ROW FORMAT SERDE 'org.apache.hadoop.hive ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hadoop.hive ql.io.SymlinkTextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://s3-bucket>/prefix/_symlink_format_manifest'
```

To add partitions to a partitioned Delta Lake table, run an ALTER TABLE ADD PARTITION command where the LOCATION parameter points to the Amazon S3 subfolder that contains the manifest for the partition.

The DDL to add partitions has the following format.

```
ALTER TABLE tbl_name
ADD IF NOT EXISTS PARTITION(pcolumn1=pvalue1[,...])
LOCATION
's3://s3-bucket/prefix/_symlink_format_manifest/partition-path'
```

Or run DDL that points directly to the Delta Lake manifest file.

```
ALTER TABLE tbl_name
ADD IF NOT EXISTS PARTITION(pcolumn1=pvalue1[,...])
LOCATION
's3://s3-bucket/prefix/_symlink_format_manifest/partition-path/manifest'
```

## Limitations and troubleshooting for Delta Lake tables

Consider the following when querying Delta Lake tables from Redshift Spectrum:

- If a manifest points to a snapshot or partition that no longer exists, queries fail until a new valid manifest has been generated. For example, this might result from a VACUUM operation on the underlying table,
- Delta Lake manifests only provide partition-level consistency.

The following table explains some potential reasons for certain errors when you query a Delta Lake table.

| Error message                                                                                          | Possible reason                                                                             |
|--------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| Delta Lake manifest in bucket <i>s3-bucket-1</i> cannot contain entries in bucket <i>s3-bucket-2</i> . | The manifest entries point to files in a different Amazon S3 bucket than the specified one. |

| Error message                                                                          | Possible reason                                                                                      |
|----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| Delta Lake files are expected to be in the same folder.                                | The manifest entries point to files that have a different Amazon S3 prefix than the specified one.   |
| File <i>filename</i> listed in Delta Lake manifest <i>manifest-path</i> was not found. | A file listed in the manifest wasn't found in Amazon S3.                                             |
| Error fetching Delta Lake manifest.                                                    | The manifest wasn't found in Amazon S3.                                                              |
| Invalid S3 Path.                                                                       | An entry in the manifest file isn't a valid Amazon S3 path, or the manifest file has been corrupted. |

## Improving Amazon Redshift Spectrum query performance

Look at the query plan to find what steps have been pushed to the Amazon Redshift Spectrum layer.

The following steps are related to the Redshift Spectrum query:

- S3 Seq Scan
- S3 HashAggregate
- S3 Query Scan
- Seq Scan PartitionInfo
- Partition Loop

The following example shows the query plan for a query that joins an external table with a local table. Note the S3 Seq Scan and S3 HashAggregate steps that were run against the data on Amazon S3.

```
explain
select top 10 spectrum.sales.eventid, sum(spectrum.sales.pricepaid)
from spectrum.sales, event
where spectrum.sales.eventid = event.eventid
and spectrum.sales.pricepaid > 30
group by spectrum.sales.eventid
order by 2 desc;
```

QUERY PLAN

```

XN Limit (cost=1001055770628.63..1001055770628.65 rows=10 width=31)

-> XN Merge (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

 Merge Key: sum(sales.derived_col2)

-> XN Network (cost=1001055770628.63..1001055770629.13 rows=200 width=31)
```

```

Send to leader

-> XN Sort (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

 Sort Key: sum(sales.derived_col2)

 -> XN HashAggregate (cost=1055770620.49..1055770620.99 rows=200
width=31)

 -> XN Hash Join DS_BCAST_INNER (cost=3119.97..1055769620.49
rows=200000 width=31)

 Hash Cond: ("outer".derived_col1 = "inner".eventid)

 -> XN S3 Query Scan sales (cost=3010.00..5010.50
rows=200000 width=31)

 -> S3 HashAggregate (cost=3010.00..3010.50
rows=200000 width=16)

 -> S3 Seq Scan spectrum.sales location:"s3://
redshift-downloads/ticket/spectrum/sales" format:TEXT (cost=0.00..2150.00 rows=172000
width=16)

 Filter: (pricepaid > 30.00)

 -> XN Hash (cost=87.98..87.98 rows=8798 width=4)

 -> XN Seq Scan on event (cost=0.00..87.98 rows=8798
width=4)

```

Note the following elements in the query plan:

- The S3 Seq Scan node shows the filter `pricepaid > 30.00` was processed in the Redshift Spectrum layer.  
A filter node under the XN S3 Query Scan node indicates predicate processing in Amazon Redshift on top of the data returned from the Redshift Spectrum layer.
- The S3 HashAggregate node indicates aggregation in the Redshift Spectrum layer for the group by clause (`group by spectrum.sales.eventid`).

Following are ways to improve Redshift Spectrum performance:

- Use Apache Parquet formatted data files. Parquet stores data in a columnar format, so Redshift Spectrum can eliminate unneeded columns from the scan. When data is in text-file format, Redshift Spectrum needs to scan the entire file.
- Use the fewest columns possible in your queries.
- Use multiple files to optimize for parallel processing. Keep your file sizes larger than 64 MB. Avoid data size skew by keeping files about the same size.
- Put your large fact tables in Amazon S3 and keep your frequently used, smaller dimension tables in your local Amazon Redshift database.
- Update external table statistics by setting the TABLE PROPERTIES numRows parameter. Use [CREATE EXTERNAL TABLE \(p. 768\)](#) or [ALTER TABLE \(p. 640\)](#) to set the TABLE PROPERTIES numRows parameter to reflect the number of rows in the table. Amazon Redshift doesn't analyze external tables to generate the table statistics that the query optimizer uses to generate a query plan. If table

statistics aren't set for an external table, Amazon Redshift generates a query execution plan. Amazon Redshift generates this plan based on the assumption that external tables are the larger tables and local tables are the smaller tables.

- The Amazon Redshift query planner pushes predicates and aggregations to the Redshift Spectrum query layer whenever possible. When large amounts of data are returned from Amazon S3, the processing is limited by your cluster's resources. Redshift Spectrum scales automatically to process large requests. Thus, your overall performance improves whenever you can push processing to the Redshift Spectrum layer.
- Write your queries to use filters and aggregations that are eligible to be pushed to the Redshift Spectrum layer.

The following are examples of some operations that can be pushed to the Redshift Spectrum layer:

- GROUP BY clauses
- Comparison conditions and pattern-matching conditions, such as LIKE.
- Aggregate functions, such as COUNT, SUM, AVG, MIN, and MAX.
- String functions.

Operations that can't be pushed to the Redshift Spectrum layer include DISTINCT and ORDER BY.

- Use partitions to limit the data that is scanned. Partition your data based on your most common query predicates, then prune partitions by filtering on partition columns. For more information, see [Partitioning Redshift Spectrum external tables \(p. 279\)](#).

Query [SVL\\_S3PARTITION \(p. 1651\)](#) to view total partitions and qualified partitions.

## Setting data handling options

You can set table parameters when you create external tables to tailor the data being queried in external tables. Otherwise, scan errors can occur. For more information, see TABLE PROPERTIES in [CREATE EXTERNAL TABLE \(p. 768\)](#). For examples, see [Data handling examples \(p. 786\)](#). For a list of errors, see [SVL\\_SPECTRUM\\_SCAN\\_ERROR \(p. 1659\)](#).

You can set the following TABLE PROPERTIES when you create external tables to specify input handling for data being queried in external tables.

- column\_count\_mismatch\_handling to identify if the file contains less or more values for a row than the number of columns specified in the external table definition.
- invalid\_char\_handling to specify input handling for invalid characters in columns containing VARCHAR, CHAR, and string data. When you specify REPLACE for invalid\_char\_handling, you can specify the replacement character to use.
- numeric\_overflow\_handling to specify cast overflow handling in columns containing integer and decimal data.
- surplus\_bytes\_handling to specify input handling for surplus bytes in columns containing VARBYTE data.
- surplus\_char\_handling to specify input handling for surplus characters in columns containing VARCHAR, CHAR, and string data.

You can set a configuration option to cancel queries that exceed a maximum number of errors. For more information, see [spectrum\\_query\\_maxerror \(p. 1698\)](#).

## Example: Performing correlated subqueries in Redshift Spectrum

You can perform correlated subqueries in Redshift Spectrum. The \$spectrum\_oid pseudocolumn provides the ability to perform correlated queries with Redshift Spectrum. To perform a correlated subquery, the pseudocolumn \$spectrum\_oid must be enabled but doesn't appear in the SQL statement. For more information, see [Pseudocolumns \(p. 278\)](#).

To create the external schema and external tables for this example, see [Getting started with Amazon Redshift Spectrum \(p. 250\)](#).

Following is an example of a correlated subquery in Redshift Spectrum.

```
select *
from myspectrum_schema.sales s
where exists
(select *
from myspectrum_schema.listing l
where l.listid = s.listid)
order by salesid
limit 5;
```

| salesid         | listid              | sellerid | buyerid | eventid | dateid | qtysold | pricepaid | commission |
|-----------------|---------------------|----------|---------|---------|--------|---------|-----------|------------|
| <i>saletime</i> |                     |          |         |         |        |         |           |            |
| 1               | 1                   | 36861    | 21191   | 7872    | 1875   | 4       | 728       | 109.2      |
|                 | 2008-02-18 02:36:48 |          |         |         |        |         |           |            |
| 2               | 4                   | 8117     | 11498   | 4337    | 1983   | 2       | 76        | 11.4       |
|                 | 2008-06-06 05:00:16 |          |         |         |        |         |           |            |
| 3               | 5                   | 1616     | 17433   | 8647    | 1983   | 2       | 350       | 52.5       |
|                 | 2008-06-06 08:26:17 |          |         |         |        |         |           |            |
| 4               | 5                   | 1616     | 19715   | 8647    | 1986   | 1       | 175       | 26.25      |
|                 | 2008-06-09 08:38:52 |          |         |         |        |         |           |            |
| 5               | 6                   | 47402    | 14115   | 8240    | 2069   | 2       | 154       | 23.1       |
|                 | 2008-08-31 09:17:02 |          |         |         |        |         |           |            |

## Monitoring metrics in Amazon Redshift Spectrum

You can monitor Amazon Redshift Spectrum queries using the following system views:

- [SVL\\_S3QUERY \(p. 1653\)](#)

Use the SVL\_S3QUERY view to get details about Redshift Spectrum queries (S3 queries) at the segment and node slice level.

- [SVL\\_S3QUERY\\_SUMMARY \(p. 1655\)](#)

Use the SVL\_S3QUERY\_SUMMARY view to get a summary of all Amazon Redshift Spectrum queries (S3 queries) that have been run on the system.

The following are some things to look for in SVL\_S3QUERY\_SUMMARY:

- The number of files that were processed by the Redshift Spectrum query.
- The number of bytes scanned from Amazon S3. The cost of a Redshift Spectrum query is reflected in the amount of data scanned from Amazon S3.

- The number of bytes returned from the Redshift Spectrum layer to the cluster. A large amount of data returned might affect system performance.
- The maximum duration and average duration of Redshift Spectrum requests. Long-running requests might indicate a bottleneck.

## Troubleshooting queries in Amazon Redshift Spectrum

Following, you can find a quick reference that identifies and addresses some common issues you might encounter with Amazon Redshift Spectrum queries. To view errors generated by Redshift Spectrum queries, query the [SVL\\_S3LOG \(p. 1650\)](#) system table.

### Topics

- [Retries exceeded \(p. 291\)](#)
- [Access throttled \(p. 291\)](#)
- [Resource limit exceeded \(p. 292\)](#)
- [No rows returned for a partitioned table \(p. 293\)](#)
- [Not authorized error \(p. 293\)](#)
- [Incompatible data formats \(p. 293\)](#)
- [Syntax error when using Hive DDL in Amazon Redshift \(p. 293\)](#)
- [Permission to create temporary tables \(p. 294\)](#)
- [Invalid range \(p. 294\)](#)
- [Invalid Parquet version number \(p. 294\)](#)

## Retries exceeded

If an Amazon Redshift Spectrum request times out, the request is canceled and resubmitted. After five failed retries, the query fails with the following error.

```
error: Spectrum Scan Error: Retries exceeded
```

Possible causes include the following:

- Large file sizes (greater than 1 GB). Check your file sizes in Amazon S3 and look for large files and file size skew. Break up large files into smaller files, between 100 MB and 1 GB. Try to make files about the same size.
- Slow network throughput. Try your query later.

## Access throttled

Amazon Redshift Spectrum is subject to the service quotas of other AWS services. Under high usage, Redshift Spectrum requests might be required to slow down, resulting in the following error.

```
error: Spectrum Scan Error: Access throttled
```

Two types of throttling can happen:

- Access throttled by Amazon S3.
- Access throttled by AWS KMS.

The error context provides more details about the type of throttling. Following, you can find causes and possible resolutions for this throttling.

## Access throttled by Amazon S3

Amazon S3 might throttle a Redshift Spectrum request if the read request rate on a [prefix](#) is too high. For information about a GET/HEAD request rate that you can achieve in Amazon S3, see [Optimizing Amazon S3 Performance](#) in *Amazon Simple Storage Service User Guide*. The Amazon S3 GET/HEAD request rate takes into account all GET/HEAD requests on a prefix so different applications accessing the same prefix share the total requests rate.

If your Redshift Spectrum requests frequently get throttled by Amazon S3, reduce the number of Amazon S3 GET/HEAD requests that Redshift Spectrum makes to Amazon S3. To do this, try merging small files into larger files. We recommend using file sizes of 64 MB or larger.

Also consider partitioning your Redshift Spectrum tables to benefit from early filtering and to reduce the number of files accessed in Amazon S3. For more information, see [Partitioning Redshift Spectrum external tables \(p. 279\)](#).

## Access throttled by AWS KMS

If you store your data in Amazon S3 using server-side encryption (SSE-S3 or SSE-KMS), Amazon S3 calls an API operation to AWS KMS for each file that Redshift Spectrum accesses. These requests count toward your cryptographic operations quota; for more information, see [AWS KMS Request Quotas](#). For more information on SSE-S3 and SSE-KMS, see [Protecting Data Using Server-Side Encryption](#) and [Protecting Data Using Server-Side Encryption with KMS keys Stored in AWS KMS](#) in *Amazon Simple Storage Service User Guide*.

A first step to reduce the number of requests that Redshift Spectrum makes to AWS KMS is to reduce the number of files accessed. To do this, try merging small files into larger files. We recommend using file sizes of 64 MB or larger.

If your Redshift Spectrum requests frequently get throttled by AWS KMS, consider requesting a quota increase for your AWS KMS request rate for cryptographic operations. To request a quota increase, see [AWS Service Limits](#) in the *Amazon Web Services General Reference*.

## Resource limit exceeded

Redshift Spectrum enforces an upper bound on the amount of memory a request can use. A Redshift Spectrum request that requires more memory fails, resulting in the following error.

```
error: Spectrum Scan Error: Resource limit exceeded
```

There are two common reasons that can cause a Redshift Spectrum request to overrun its memory allowance:

- Redshift Spectrum processes a large chunk of data that can't be split in smaller chunks.
- A large aggregation step is processed by Redshift Spectrum.

We recommend using a file format that supports parallel reads with split sizes of 128 MB or less. See [Creating data files for queries in Amazon Redshift Spectrum \(p. 270\)](#) for supported file formats and

generic guidelines for data file creation. When using file formats or compression algorithms that don't support parallel reads, we recommend keeping file sizes between 64 MB and 128 MB.

## No rows returned for a partitioned table

If your query returns zero rows from a partitioned external table, check whether a partition has been added for this external table. Redshift Spectrum only scans files in an Amazon S3 location that has been explicitly added using `ALTER TABLE ... ADD PARTITION`. Query the [SVV\\_EXTERNAL\\_PARTITIONS \(p. 1429\)](#) view to find existing partitions. Run `ALTER TABLE ... ADD PARTITION` for each missing partition.

## Not authorized error

Verify that the IAM role for the cluster allows access to the Amazon S3 file objects. If your external database is on Amazon Athena, verify that the IAM role allows access to Athena resources. For more information, see [IAM policies for Amazon Redshift Spectrum \(p. 259\)](#).

## Incompatible data formats

For a columnar file format, such as Apache Parquet, the column type is embedded with the data. The column type in the `CREATE EXTERNAL TABLE` definition must match the column type of the data file. If there is a mismatch, you receive an error similar to the following:

```
File 'https://s3bucket/location/file' has an incompatible Parquet schema
for column 's3://s3bucket/location.col1'. Column type: VARCHAR, Par
```

The error message might be truncated due to the limit on message length. To retrieve the complete error message, including column name and column type, query the [SVL\\_S3LOG \(p. 1650\)](#) system view.

The following example queries SVL\_S3LOG for the last query completed.

```
select message
from svl_s3log
where query = pg_last_query_id()
order by query,segment,slice;
```

The following is an example of a result that shows the full error message.

| message                                                                                                                                                                                                               |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Spectrum Scan Error. File 'https://s3bucket/location/file' has an incompatible Parquet schema for column 's3bucket/location.col1'.<br>Column type: VARCHAR, Parquet schema:\optional int64 l_orderkey [i:0 d:1 r:0]\n |

To correct the error, alter the external table to match the column type of the Parquet file.

## Syntax error when using Hive DDL in Amazon Redshift

Amazon Redshift supports data definition language (DDL) for `CREATE EXTERNAL TABLE` that is similar to Hive DDL. However, the two types of DDL aren't always exactly the same. If you copy Hive DDL to create or alter Amazon Redshift external tables, you might encounter syntax errors. The following are examples of differences between Amazon Redshift and Hive DDL:

- Amazon Redshift requires single quotation marks ('') where Hive DDL supports double quotation marks ("").
- Amazon Redshift doesn't support the STRING data type. Use VARCHAR instead.

## Permission to create temporary tables

To run Redshift Spectrum queries, the database user must have permission to create temporary tables in the database. The following example grants temporary permission on the database spectrumdb to the spectrumusers user group.

```
grant temp on database spectrumdb to group spectrumusers;
```

For more information, see [GRANT \(p. 887\)](#).

## Invalid range

Redshift Spectrum expects that files in Amazon S3 that belong to an external table are not overwritten during a query. If this happens, it can result in the following error.

```
Error: HTTP response error code: 416 Message: InvalidRange The requested range is not satisfiable
```

To avoid the error, make sure Amazon S3 files are not overwritten while they are queried with Redshift Spectrum.

## Invalid Parquet version number

Redshift Spectrum checks the metadata of each Apache Parquet file it accesses. If the check fails, it can result in an error similar to the following:

```
File 'https://s3.region.amazonaws.com/s3bucket/location/file has an invalid version number
```

There are two common reasons that can cause the check to fail:

- The Parquet file has been overwritten during the query (see [Invalid range \(p. 294\)](#)).
- The Parquet file is corrupt.

# Tutorial: Querying nested data with Amazon Redshift Spectrum

## Overview

Amazon Redshift Spectrum supports querying nested data in Parquet, ORC, JSON, and Ion file formats. Redshift Spectrum accesses the data using external tables. You can create external tables that use the complex data types struct, array, and map.

For example, suppose that your data file contains the following data in Amazon S3 in a folder named customers. Although there isn't a single root element, each JSON object in this sample data represents a row in a table.

```
{"id": 1,
 "name": {"given": "John", "family": "Smith"},
 "phones": ["123-457789"],
 "orders": [{"shipdate": "2018-03-01T11:59:59.000Z", "price": 100.50},
 {"shipdate": "2018-03-01T09:10:00.000Z", "price": 99.12}]
}
{"id": 2,
 "name": {"given": "Jenny", "family": "Doe"},
 "phones": ["858-8675309", "415-9876543"],
 "orders": []
}
{"id": 3,
 "name": {"given": "Andy", "family": "Jones"},
 "phones": [],
 "orders": [{"shipdate": "2018-03-02T08:02:15.000Z", "price": 13.50}]
}
```

You can use Redshift Spectrum to query this data. The following tutorial shows you how to do so.

For tutorial prerequisites, steps, and nested data use cases, see the following topics:

- [Prerequisites \(p. 295\)](#)
- [Step 1: Create an external table that contains nested data \(p. 295\)](#)
- [Step 2: Query your nested data in Amazon S3 with SQL extensions \(p. 296\)](#)
- [Nested data use cases \(p. 300\)](#)
- [Nested data limitations \(p. 301\)](#)
- [Serializing complex nested JSON \(p. 302\)](#)

## Prerequisites

If you are not using Redshift Spectrum yet, follow the steps in the [Getting started with Amazon Redshift Spectrum \(p. 250\)](#) before continuing.

To create an external schema, replace the IAM role ARN in the following command with the role ARN you created in [Create an IAM role \(p. 251\)](#). Then run the command in your SQL client.

```
create external schema spectrum
from data catalog
database 'myspectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myspectrum_role'
create external database if not exists;
```

## Step 1: Create an external table that contains nested data

You can view the [source data](#) by downloading it from Amazon S3.

To create the external table for this tutorial, run the following command.

```
CREATE EXTERNAL TABLE spectrum.customers (
 id int,
 name struct<given:varchar(20), family:varchar(20)>,
 phones array<varchar(20)>,
```

```
orders array<struct<shipdate:timestamp, price:double precision>>
)
STORED AS PARQUET
LOCATION 's3://redshift-downloads/ticket/spectrum/customers/';
```

In the example preceding, the external table spectrum.customers uses the struct and array data types to define columns with nested data. Amazon Redshift Spectrum supports querying nested data in Parquet, ORC, JSON, and Ion file formats. The LOCATION parameter has to refer to the Amazon S3 folder that contains the nested data or files.

**Note**

Amazon Redshift doesn't support complex data types in an Amazon Redshift database table. You can use complex data types only with Redshift Spectrum external tables.

You can nest array and struct types at any level. For example, you can define a column named toparray as shown in the following example.

```
toparray array<struct<nestedarray:
 array<struct<morenestedarray:
 array<string>>>>
```

You can also nest struct types as shown for column x in the following example.

```
x struct<a: string,
 b: struct<c: integer,
 d: struct<e: string>
 >
 >
```

## Step 2: Query your nested data in Amazon S3 with SQL extensions

Redshift Spectrum supports querying array, map, and struct complex types through extensions to the Amazon Redshift SQL syntax.

### Extension 1: Access to columns of structs

You can extract data from struct columns using a dot notation that concatenates field names into paths. For example, the following query returns given and family names for customers. The given name is accessed by the long path c.name.given. The family name is accessed by the long path c.name.family.

```
SELECT c.id, c.name.given, c.name.family
FROM spectrum.customers c;
```

The preceding query returns the following data.

| id       | given | family |
|----------|-------|--------|
| 1        | John  | Smith  |
| 2        | Jenny | Doe    |
| 3        | Andy  | Jones  |
| (3 rows) |       |        |

A struct can be a column of another struct, which can be a column of another struct, at any level. The paths that access columns in such deeply nested structs can be arbitrarily long. For example, see the definition for the column x in the following example.

```
x struct<a: string,
 b: struct<c: integer,
 d: struct<e: string>
 >
 >
```

You can access the data in e as x.b.d.e.

**Note**

You use structs only to describe the path to the fields that they contain. You can't access them directly in a query or return them from a query.

## Extension 2: Ranging over arrays in a FROM clause

You can extract data from array columns (and, by extension, map columns) by specifying the array columns in a FROM clause in place of table names. The extension applies to the FROM clause of the main query, and also the FROM clauses of subqueries. You can't reference array elements by position, such as c.orders[0].

By combining ranging over arrays with joins, you can achieve various kinds of unnesting, as explained in the following use cases.

### Unnesting using inner joins

The following query selects customer IDs and order ship dates for customers that have orders. The SQL extension in the FROM clause c.orders o depends on the alias c.

```
SELECT c.id, o.shipdate
FROM spectrum.customers c, c.orders o
```

For each customer c that has orders, the FROM clause returns one row for each order o of the customer c. That row combines the customer row c and the order row o. Then the SELECT clause keeps only the c.id and o.shipdate. The result is the following.

| id       | shipdate            |
|----------|---------------------|
| 1        | 2018-03-01 11:59:59 |
| 1        | 2018-03-01 09:10:00 |
| 3        | 2018-03-02 08:02:15 |
| (3 rows) |                     |

The alias `c` provides access to the customer fields, and the alias `o` provides access to the order fields.

The semantics are similar to standard SQL. You can think of the `FROM` clause as running the following nested loop, which is followed by `SELECT` choosing the fields to output.

```
for each customer c in spectrum.customers
 for each order o in c.orders
 output c.id and o.shipdate
```

Therefore, if a customer doesn't have an order, the customer doesn't appear in the result.

You can also think of this as the `FROM` clause performing a `JOIN` with the `customers` table and the `orders` array. In fact, you can also write the query as shown in the following example.

```
SELECT c.id, o.shipdate
FROM spectrum.customers c INNER JOIN c.orders o ON true
```

#### Note

If a schema named `c` exists with a table named `orders`, then `c.orders` refers to the table `orders`, and not the array column of `customers`.

### Unnesting using left joins

The following query outputs all customer names and their orders. If a customer hasn't placed an order, the customer's name is still returned. However, in this case the order columns are `NULL`, as shown in the following example for Jenny Doe.

```
SELECT c.id, c.name.given, c.name.family, o.shipdate, o.price
FROM spectrum.customers c LEFT JOIN c.orders o ON true
```

The preceding query returns the following data.

| id | given | family | shipdate            | price |
|----|-------|--------|---------------------|-------|
| 1  | John  | Smith  | 2018-03-01 11:59:59 | 100.5 |
| 1  | John  | Smith  | 2018-03-01 09:10:00 | 99.12 |
| 2  | Jenny | Doe    |                     |       |
| 3  | Andy  | Jones  | 2018-03-02 08:02:15 | 13.5  |

(4 rows)

### Extension 3: Accessing an array of scalars directly using an alias

When an alias `p` in a `FROM` clause ranges over an array of scalars, the query refers to the values of `p` simply as `p`. For example, the following query produces pairs of customer names and phone numbers.

```
SELECT c.name.given, c.name.family, p AS phone
FROM spectrum.customers c LEFT JOIN c.phones p ON true
```

The preceding query returns the following data.

| given    | family | phone       |
|----------|--------|-------------|
| John     | Smith  | 123-4577891 |
| Jenny    | Doe    | 858-8675309 |
| Jenny    | Doe    | 415-9876543 |
| Andy     | Jones  |             |
| (4 rows) |        |             |

## Extension 4: Accessing elements of maps

Redshift Spectrum treats the map data type as an array type that contains struct types with a key column and a value column. The key must be a scalar; the value can be any data type.

For example, the following code creates an external table with a map for storing phone numbers.

```
CREATE EXTERNAL TABLE spectrum.customers2 (
 id int,
 name struct<given:varchar(20), family:varchar(20)>,
 phones map<varchar(20), varchar(20)>,
 orders array<struct<shipdate:timestamp, price:double precision>>
)
STORED AS PARQUET
LOCATION 's3://redshift-downloads/ticket/spectrum/customers/';
```

Because a map type behaves like an array type with columns key and value, you can think of the preceding schemas as if they were the following.

```
CREATE EXTERNAL TABLE spectrum.customers3 (
 id int,
 name struct<given:varchar(20), family:varchar(20)>,
 phones array<struct<key:varchar(20), value:varchar(20)>>,
 orders array<struct<shipdate:timestamp, price:double precision>>
)
STORED AS PARQUET
LOCATION 's3://redshift-downloads/ticket/spectrum/customers/';
```

The following query returns the names of customers with a mobile phone number and returns the number for each name. The map query is treated as the equivalent of querying a nested array of struct types. The following query only returns data if you have created the external table as described previously.

```
SELECT c.name.given, c.name.family, p.value
FROM spectrum.customers c, c.phones p
WHERE p.key = 'mobile'
```

### Note

The key for a map is a string for Ion and JSON file types.

## Nested data use cases

You can combine the extensions described previously with the usual SQL features. The following use cases illustrate some common combinations. These examples help demonstrate how you can use nested data. They aren't part of the tutorial.

### Topics

- [Ingesting nested data \(p. 300\)](#)
- [Aggregating nested data with subqueries \(p. 300\)](#)
- [Joining Amazon Redshift and nested data \(p. 301\)](#)

## Ingesting nested data

You can use a `CREATE TABLE AS` statement to ingest data from an external table that contains complex data types. The following query extracts all customers and their phone numbers from the external table, using `LEFT JOIN`, and stores them in the Amazon Redshift table `CustomerPhones`.

```
CREATE TABLE CustomerPhones AS
SELECT c.name.given, c.name.family, p AS phone
FROM spectrum.customers c LEFT JOIN c.phones p ON true
```

## Aggregating nested data with subqueries

You can use a subquery to aggregate nested data. The following example illustrates this approach.

```
SELECT c.name.given, c.name.family, (SELECT COUNT(*) FROM c.orders o) AS ordercount
FROM spectrum.customers c
```

The following data is returned.

| given    | family | ordercount |
|----------|--------|------------|
| Jenny    | Doe    | 0          |
| John     | Smith  | 2          |
| Andy     | Jones  | 1          |
| (3 rows) |        |            |

### Note

When you aggregate nested data by grouping by the parent row, the most efficient way is the one shown in the previous example. In that example, the nested rows of `c.orders` are grouped by their parent row `c`. Alternatively, if you know that `id` is unique for each customer and `o.shipdate` is never null, you can aggregate as shown in the following example. However, this approach generally isn't as efficient as the previous example.

```
SELECT c.name.given, c.name.family, COUNT(o.shipdate) AS ordercount
FROM spectrum.customers c LEFT JOIN c.orders o ON true
GROUP BY c.id, c.name.given, c.name.family
```

You can also write the query by using a subquery in the FROM clause that refers to an alias (s) of the ancestor query and extracts array data. The following example demonstrates this approach.

```
SELECT c.name.given, c.name.family, s.count AS ordercount
FROM spectrum.customers c, (SELECT count(*) AS count FROM c.orders o) s
```

## Joining Amazon Redshift and nested data

You can also join Amazon Redshift data with nested data in an external table. For example, suppose that you have the following nested data in Amazon S3.

```
CREATE EXTERNAL TABLE spectrum.customers2 (
 id int,
 name struct<given:varchar(20), family:varchar(20)>,
 phones array<varchar(20)>,
 orders array<struct<shipdate:timestamp, item:int>>
)
```

Suppose also that you have the following table in Amazon Redshift.

```
CREATE TABLE prices (
 id int,
 price double precision
)
```

The following query finds the total number and amount of each customer's purchases based on the preceding. The following example is only an illustration. It only returns data if you have created the tables described previously.

```
SELECT c.name.given, c.name.family, COUNT(o.date) AS ordercount, SUM(p.price) AS
 ordersum
FROM spectrum.customers2 c, c.orders o, prices p ON o.item = p.id
GROUP BY c.id, c.name.given, c.name.family
```

## Nested data limitations

The following limitations apply to nested data:

- An array can only contain scalars or struct types. Array types can't contain array or map types.
- Redshift Spectrum supports complex data types only as external tables.
- Query and subquery result columns must be scalar.
- If an OUTER JOIN expression refers to a nested table, it can refer only to that table and its nested arrays (and maps). If an OUTER JOIN expression doesn't refer to a nested table, it can refer to any number of non-nested tables.

- If a FROM clause in a subquery refers to a nested table, it can't refer to any other table.
- If a subquery depends on a nested table that refers to a parent, you can use the parent only in the FROM clause. You can't use the parent in any other clauses, such as a SELECT or WHERE clause. For example, the following query isn't run.

```
SELECT c.name.given
FROM spectrum.customers c
WHERE (SELECT COUNT(c.id) FROM c.phones p WHERE p LIKE '858%') > 1
```

The following query works because the parent c is used only in the FROM clause of the subquery.

```
SELECT c.name.given
FROM spectrum.customers c
WHERE (SELECT COUNT(*) FROM c.phones p WHERE p LIKE '858%') > 1
```

- A subquery that accesses nested data anywhere other than the FROM clause must return a single value. The only exceptions are (NOT) EXISTS operators in a WHERE clause.
- (NOT) IN is not supported.
- The maximum nesting depth for all nested types is 100. This restriction applies to all file formats (Parquet, ORC, Ion, and JSON).
- Aggregation subqueries that access nested data can only refer to arrays and maps in their FROM clause, not to an external table.
- Querying the pseudocolumns of nested data in a Redshift Spectrum table is not supported. For more information, see [Pseudocolumns \(p. 278\)](#).

## Serializing complex nested JSON

An alternate to methods demonstrated in this tutorial is to query top-level nested collection columns as serialized JSON. You can use the serialization to inspect, convert, and ingest nested data as JSON with Redshift Spectrum. This method is supported for ORC, JSON, Ion, and Parquet formats. Use the session configuration parameter json\_serialization\_enable to configure the serialization behavior. When set, complex JSON data types are serialized to VARCHAR(65535). The nested JSON can be accessed with [JSON functions \(p. 1142\)](#). For more information, see [json\\_serialization\\_enable \(p. 1693\)](#).

For example, without setting json\_serialization\_enable, the following queries that access nested columns directly fail.

```
SELECT * FROM spectrum.customers LIMIT 1;
=> ERROR: Nested tables do not support '*' in the SELECT clause.

SELECT name FROM spectrum.customers LIMIT 1;
=> ERROR: column "name" does not exist in customers
```

Setting json\_serialization\_enable enables querying top-level collections directly.

```
SET json_serialization_enable TO true;
```

```

SELECT * FROM spectrum.customers order by id LIMIT 1;

id | name | phones | orders
---+-----+-----+-----+
1 | {"given": "John", "family": "Smith"} | ["123-457789"] | [{"shipdate": "2018-03-01T11:59:59.000Z", "price": 100.50}, {"shipdate": "2018-03-01T09:10:00.000Z", "price": 99.12}]

SELECT name FROM spectrum.customers order by id LIMIT 1;

name

{"given": "John", "family": "Smith"}

```

Consider the following items when serializing nested JSON.

- When collection columns are serialized as VARCHAR(65535), their nested subfields can't be accessed directly as part of the query syntax (for example, in the filter clause). However, JSON functions can be used to access nested JSON.
- The following specialized representations are not supported:
  - ORC unions
  - ORC maps with complex type keys
  - Ion datagrams
  - Ion SEXP
- Timestamps are returned as ISO serialized strings.
- Primitive map keys are promoted to string (for example, 1 to "1").
- Top-level null values are serialized as NULLs.
- If the serialization overflows the maximum VARCHAR size of 65535, the cell is set to NULL.

## Serializing complex types containing JSON strings

By default, string values contained in nested collections are serialized as escaped JSON strings. Escaping might be undesirable when the strings are valid JSON. Instead you might want to write nested subelements or fields that are VARCHAR directly as JSON. Enable this behavior with the json\_serialization\_parse\_nested\_strings session-level configuration. When both json\_serialization\_enable and json\_serialization\_parse\_nested\_strings are set, valid JSON values are serialized inline without escape characters. When the value is not valid JSON, it is escaped as if the json\_serialization\_parse\_nested\_strings configuration value was not set. For more information, see [json\\_serialization\\_parse\\_nested\\_strings \(p. 1693\)](#).

For example, assume the data from the previous example contained JSON as a structs complex type in the name VARCHAR(20) field:

```

name

>{"given": "{\"first\": \"John\", \"middle\": \"James\"}", "family": "Smith"}

```

When json\_serialization\_parse\_nested\_strings is set, the name column is serialized as follows:

```

SET json_serialization_enable TO true;
SET json_serialization_parse_nested_strings TO true;

```

```
SELECT name FROM spectrum.customers order by id LIMIT 1;

name

{"given": {"first": "John", "middle": "James"}, "family": "Smith"}
```

Instead of being escaped like this:

```
SET json_serialization_enable TO true;
SELECT name FROM spectrum.customers order by id LIMIT 1;

name

{"given": "{\"first\": \"John\", \"middle\": \"James\"}", "family": "Smith"}
```

# Using HyperLogLog sketches in Amazon Redshift

*HyperLogLog* is an algorithm used for estimating the cardinality of a multiset. *Cardinality* refers to the number of distinct values in a multiset. For example, in the set of {4,3,6,2,2,6,4,3,6,2,2,3}, the cardinality is 4 with distinct values of 4, 3, 6, and 2.

The precision of the HyperLogLog algorithm (also known as m value) can affect the accuracy of the estimated cardinality. During the cardinality estimation, Amazon Redshift uses a default precision value of 15. This value can be up to 26 for smaller datasets. Thus, the average relative error ranges between 0.01–0.6%.

When calculating the cardinality of a multiset, the HyperLogLog algorithm generates a construct called an HLL sketch. An *HLL sketch* encapsulates information about the distinct values in a multiset. The Amazon Redshift data type `HLLSKETCH` represents such sketch values. This data type can be used to store sketches in an Amazon Redshift table. Additionally, Amazon Redshift supports operations that can be applied to `HLLSKETCH` values as aggregate and scalar functions. You can use these functions to extract the cardinality of an `HLLSKETCH` and combine multiple `HLLSKETCH` values.

The `HLLSKETCH` data type offers significant query performance benefits when extracting the cardinality from large datasets. You can preaggregate these datasets using `HLLSKETCH` values and store them in tables. Amazon Redshift can extract the cardinality directly from the stored `HLLSKETCH` values without accessing the underlying datasets.

When processing HLL sketches, Amazon Redshift performs optimizations that minimize the memory footprint of the sketch and maximize the precision of the extracted cardinality. Amazon Redshift uses two representations for HLL sketches, sparse and dense. An `HLLSKETCH` starts in sparse format. As new values are inserted into it, its size increases. After its size reaches the size of the dense representation, Amazon Redshift automatically converts the sketch from sparse to dense.

Amazon Redshift imports, exports, and prints an `HLLSKETCH` as JSON when the sketch is in a sparse format. Amazon Redshift imports, exports, and prints an `HLLSKETCH` as a Base64 string when the sketch is in a dense format. For more information about UNLOAD, see [Unloading the `HLLSKETCH` data type \(p. 998\)](#). To import text or comma-separated value (CSV) data into Amazon Redshift, use the COPY command. For more information, see [Loading the `HLLSKETCH` data type \(p. 718\)](#).

For information about functions used with HyperLogLog, see [HyperLogLog functions \(p. 1138\)](#).

## Topics

- [Considerations \(p. 305\)](#)
- [Limitations \(p. 306\)](#)
- [Examples \(p. 306\)](#)

## Considerations

The following are considerations for using HyperLogLog in Amazon Redshift:

- The following non-HyperLogLog functions can accept an input of type `HLLSKETCH` or columns of type `HLLSKETCH`:

- The aggregate function COUNT
- The conditional expressions COALESCE and NVL
- CASE expressions
- The supported encoding is RAW.
- You can perform an UNLOAD operation on table with HLLSKETCH columns into text or CSV. You can use the UNLOAD HLLSKETCH columns to write HLLSKETCH data. Amazon Redshift shows the data in a JSON format for a sparse representation or a Base64 format for a dense representation. For more information about UNLOAD, see [Unloading the HLLSKETCH data type \(p. 998\)](#).

The following shows the format used for a sparse HyperLogLog sketch represented in a JSON format.

```
{"version":1,"logm":15,"sparse":{"indices": [15099259,33107846,37891580,50065963],"values":[2,3,2,1]}}
```

- You can import text or CSV data into Amazon Redshift using the COPY command. For more information, see [Loading the HLLSKETCH data type \(p. 718\)](#).
- The default encoding for HLLSKETCH is RAW. For more information, see [Compression encodings \(p. 39\)](#).

## Limitations

The following are limitations for using HyperLogLog in Amazon Redshift:

- Amazon Redshift tables don't support an HLLSKETCH column as a sort key or a distribution key for an Amazon Redshift table.
- Amazon Redshift doesn't support HLLSKETCH columns in ORDER BY, GROUP BY, or DISTINCT clauses.
- You can only UNLOAD HLLSKETCH columns to text or CSV format. Amazon Redshift then writes the HLLSKETCH data in either a JSON format or a Base64 format. For more information about UNLOAD, see [UNLOAD \(p. 991\)](#).
- Amazon Redshift only supports HyperLogLog sketches with a precision (logm value) of 15.
- JDBC and ODBC drivers don't support the HLLSKETCH data type. Therefore, the result set uses VARCHAR to represent the HLLSKETCH values.
- Amazon Redshift Spectrum doesn't natively support the HLLSKETCH data. Therefore, you can't create or alter an external table with an HLLSKETCH column.
- Data types for Python user-defined functions (UDFs) don't support the HLLSKETCH data type. For more information about Python UDFs, see [Creating a scalar Python UDF \(p. 152\)](#).

## Examples

### Example: Return cardinality in a subquery

The following example returns the cardinality for each sketch in a subquery for a table named *Sales*.

```
CREATE TABLE Sales (customer VARCHAR, country VARCHAR, amount BIGINT);
INSERT INTO Sales VALUES ('David Joe', 'Greece', 14.5), ('David Joe', 'Greece', 19.95),
('John Doe', 'USA', 29.95), ('John Doe', 'USA', 19.95), ('George Spanos', 'Greece', 9.95),
('George Spanos', 'Greece', 2.95);
```

The following query generates an HLL sketch for the customers of each country and extracts the cardinality. This shows unique customers from each country.

```
SELECT hll_cardinality(sketch), country
FROM (SELECT hll_create_sketch(customer) AS sketch, country
 FROM Sales
 GROUP BY country) AS hll_subquery;

hll_cardinality | country
-----+-----
 1 | USA
 2 | Greece
...
```

## Example: Return an HLLSKETCH type from combined sketches in a subquery

The following example returns a single HLLSKETCH type that represents the combination of individual sketches from a subquery. The sketches are combined by using the HLL\_COMBINE aggregate function.

```
SELECT hll_combine(sketch)
FROM (SELECT hll_create_sketch(customers) AS sketch
 FROM Sales
 GROUP BY country) AS hll_subquery

 hll_combine

[{"version":1,"logm":15,"sparse":{"indices":[29808639,35021072,47612452],"values":[1,1,1]}]
(1 row)
```

## Example: Return a HyperLogLog sketch from combining multiple sketches

For the following example, suppose that the table page\\_users stores preaggregated sketches for each page that users visited on a given website. Each row in this table contains a HyperLogLog sketch that represents all user IDs that show the visited pages.

| page_users     |           |              |
|----------------|-----------|--------------|
| _PARTITIONTIME | page      | sketch       |
| 2019-07-28     | homepage  | CHAQkAQYA... |
| 2019-07-28     | Product A | CHAQxPnYB... |

The following example unions the preaggregated multiple sketches and generates a single sketch. This sketch encapsulates the collective cardinality that each sketch encapsulates.

```
SELECT hll_combine(sketch) as sketch
FROM page_users
```

The output looks similar to the following.

| sketch |
|--------|
|--------|

```
-- | CHAQ3sGoCxgCIAuCB4iAIBgTIBgqgIAgAwY.... |
-- +-----+-----+
```

When a new sketch is created, you can use the HLL\_CARDINALITY function to get the collective distinct values, as shown following.

```
SELECT hll_cardinality(sketch)
FROM (
 SELECT
 hll_combine(sketch) as sketch
 FROM page_users
) AS hll_subquery
```

The output looks similar to the following.

```
-- +-----+
-- | count |
-- +-----+
-- | 54356 |
-- +-----+
```

## Example: Generate HyperLogLog sketches over S3 data using external tables

The following examples cache HyperLogLog sketches to avoid directly accessing Amazon S3 for cardinality estimation.

You can preaggregate and cache HyperLogLog sketches in external tables defined to hold Amazon S3 data. By doing this, you can extract cardinality estimates without accessing the underlying base data.

For example, suppose that you have unloaded a set of tab-delimited text files into Amazon S3. You run the following query to define an external table named sales in the Amazon Redshift external schema named spectrum.

```
create external table spectrum.sales(
 sellerid integer,
 buyerid integer,
 dateid smallint,
 qtysold smallint,
 product_type integer,
 pricepaid decimal(8,2),
 commission decimal(8,2),
 saletime date)
row format delimited
fields terminated by '\t' stored as textfile
location 's3://awssampledbuswest2/ticket/spectrum/sales/';
```

Suppose that you want to compute the distinct buyers who purchased an item on arbitrary dates. To do so, the following example generates sketches for the buyer IDs for each day of the year and stores the result in the Amazon Redshift table hll\_sales.

```
CREATE TABLE hll_sales AS
SELECT saletime, hll_create_sketch(buyerid) AS sketch
FROM spectrum.sales
GROUP BY saletime;
```

The output looks similar to the following.

```
-- hll_sales
-- | saletime | sketch |
-- +-----+-----+
-- | 2018-11-23 | "CHAQKAQYA...""
-- | 2018-11-24 | "TNLMLMLKK...""
-- | 2018-11-25 | "KMNKLLOKM...""
-- | 2018-11-26 | "MMKNKLLMO...""
-- | 2018-11-27 | "MMLSKNLPM...""
-- +-----+-----+
```

The following query extracts the estimated number of distinct buyers that purchased an item during the Friday after Thanksgiving.

```
SELECT hll_cardinality(sketch) as distinct_buyers
FROM hll_sales
WHERE saletime = '2018-11-23';
```

The output looks similar to the following.

```
distinct_buyers

1771
```

Suppose that you want the number of distinct users who bought an item on a certain range of dates. An example might be from the Friday after Thanksgiving to the following Monday. To get this, the following query uses the `hll_combine` aggregate function. This function enables you to avoid double-counting buyers who purchased an item on more than one day of the selected range.

```
SELECT hll_cardinality(hll_combine(sketch)) as distinct_buyers
FROM hll_sales
WHERE saletime BETWEEN '2018-11-23' AND '2018-11-26';
```

The output looks similar to the following.

```
distinct_buyers

232152
```

To keep the `hll_sales` table up-to-date, run the following query at the end of each day. Doing this generates an HyperLogLog sketch based on the IDs of buyers that purchased an item today and adds it to the `hll_sales` table.

```
INSERT INTO hll_sales
SELECT saletime, hll_create_sketch(buyerid)
FROM spectrum.sales
WHERE saletime = to_char(now(), 'YYYY-MM-DD');
```

# Querying data across databases

By using *cross-database queries* in Amazon Redshift, you can query across databases in an Amazon Redshift cluster. With cross-database queries, you can query data from any database in the Amazon Redshift cluster, regardless of which database you are connected to. Cross-database queries eliminate data copies and simplify your data organization to support multiple business groups from the same data warehouse.

With cross-database queries, you can do the following:

- **Query data across databases in your Amazon Redshift cluster.**

Not only can you query from databases that you are connected to, you can also read from any other databases that you have permissions to.

When you query database objects on any other unconnected databases, you have read access only to those database objects. You can use cross-database queries to access data from any of the databases on your Amazon Redshift cluster without having to connect to that specific database. Doing this can help you query and join data that is spread across multiple databases in your Amazon Redshift cluster quickly and easily.

You can also join datasets from multiple databases in a single query and analyze the data using business intelligence (BI) or analytics tools. You can continue to set up granular table or column-level access controls for users by using standard Amazon Redshift SQL commands. By doing so, you can help ensure that users see only the relevant subsets of the data that they have permissions for.

- **Query objects.**

You can query other database objects using fully qualified object names expressed with the three-part notation. The full path to any database object consists of three components: database name, schema, and name of the object. You can access any object from any other database using the full path notation, `database_name.schema_name.object_name`. To access a particular column, use `database_name.schema_name.object_name.column_name`.

You can also create an alias for a schema in another database using the external schema notation. This external schema references to another database and schema pair. Query can access the other database object using the external schema notation, `external_schema_name.object_name`.

In the same read-only query, you can query various database objects, such as user tables, regular views, materialized views, and late-binding views from other databases.

- **Manage permissions.**

Users with access privileges for objects in any databases in an Amazon Redshift cluster can query those objects. You grant privileges to users and user groups using the [GRANT \(p. 887\)](#) command. You can also revoke privileges using the [REVOKE \(p. 916\)](#) command when a user no longer requires the access to specific database objects.

- **Work with metadata and BI tools.**

You can create an external schema to refer to a schema in another Amazon Redshift database within the same Amazon Redshift cluster. For information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#) command.

After external schema references are created, Amazon Redshift shows the tables under the schema of the other database in [SVV\\_EXTERNAL\\_TABLES \(p. 1431\)](#) and [SVV\\_EXTERNAL\\_COLUMNS \(p. 1428\)](#) for the tools to explore the metadata.

To integrate cross-database query with BI tools, you can use the following system views. These help you view information about the metadata of objects in the connected and other databases on the Amazon Redshift cluster.

Following are system views that show all Amazon Redshift objects and external objects of all databases in your Amazon Redshift cluster:

- [SVV\\_ALL\\_COLUMNS \(p. 1412\)](#)
- [SVV\\_ALL\\_SCHEMAS \(p. 1413\)](#)
- [SVV\\_ALL\\_TABLES \(p. 1414\)](#)

Following are system views that show all Amazon Redshift objects of all databases in your Amazon Redshift cluster:

- [SVV\\_REDSHIFT\\_COLUMNS \(p. 1444\)](#)
- [SVV\\_REDSHIFT\\_DATABASES \(p. 1445\)](#)
- [SVV\\_REDSHIFT\\_FUNCTIONS \(p. 1446\)](#)
- [SVV\\_REDSHIFT\\_SCHEMAS \(p. 1447\)](#)
- [SVV\\_REDSHIFT\\_TABLES \(p. 1448\)](#)

#### Topics

- [Considerations \(p. 311\)](#)
- [Examples of using a cross-database query \(p. 312\)](#)
- [Using cross-database queries with the query editor \(p. 315\)](#)

## Considerations

When you work with the cross-database query feature in Amazon Redshift, consider the following:

- Amazon Redshift supports cross-database query on the ra3.4xlarge, ra3.16xlarge, and ra3.xlplus node types.
- Amazon Redshift supports joining data from tables or views across one or more databases in the same Amazon Redshift cluster.
- Amazon Redshift Serverless supports the same cross-database capabilities as Amazon Redshift clusters, so you can join data from tables or views across one or more databases in a serverless namespace.
- All queries in a transaction on the connected database read data in the same state of the other database as the data was at the beginning of the transaction. This approach helps to provide query transactional consistency across databases. Amazon Redshift supports transactional consistency for cross-database queries.
- To get metadata across databases, use SVV\_ALL\* and SVV\_REDSHIFT\* metadata views. You can't use the three-part notation or external schemas to query cross-database metadata tables or views under information\_schema and pg\_catalog.

## Limitations

When you work with the cross-database query feature in Amazon Redshift, be aware of the limitations following:

- When you query database objects on any other unconnected databases, you have read access only to those database objects.

- You can't query views that are created on other databases that refer to objects of yet another database.
- You can only create late-binding and materialized views on objects of other databases in the cluster. You can't create regular views on objects of other databases in the cluster.
- Amazon Redshift doesn't support tables with column-level privileges for cross-database queries.
- Amazon Redshift doesn't support concurrency scaling for the queries that read data from other databases.
- Amazon Redshift doesn't support query catalog objects on AWS Glue or federated databases. To query these objects, first create external schemas that refer to those external data sources in each database.
- Amazon Redshift doesn't support result cache for cross-database queries.

## Examples of using a cross-database query

Use the following examples to help learn how to set up a cross-database query that references an Amazon Redshift database.

To start, create databases db1 and db2 and users user1 and user2 in your Amazon Redshift cluster. For more information, see [CREATE DATABASE \(p. 745\)](#) and [CREATE USER \(p. 846\)](#).

```
--As user1 on db1
CREATE DATABASE db1;

CREATE DATABASE db2;

CREATE USER user1 PASSWORD 'Redshift01';

CREATE USER user2 PASSWORD 'Redshift01';
```

As user1 on db1, create a table, grant access privileges to user2, and insert values into table1. For more information, see [GRANT \(p. 887\)](#) and [INSERT \(p. 902\)](#).

```
--As user1 on db1
CREATE TABLE table1 (c1 int, c2 int, c3 int);

GRANT SELECT ON table1 TO user2;

INSERT INTO table1 VALUES (1,2,3),(4,5,6),(7,8,9);
```

As user2 on db2, run a cross-database query in db2 using the three-part notation.

```
--As user2 on db2
SELECT * from db1.public.table1 ORDER BY c1;
c1 | c2 | c3
----+---+---
1 | 2 | 3
4 | 5 | 6
7 | 8 | 9
(3 rows)
```

As user2 on db2, create an external schema and run a cross-database query in db2 using the external schema notation.

```
--As user2 on db2
CREATE EXTERNAL SCHEMA db1_public_sch
```

```
FROM REDSHIFT DATABASE 'db1' SCHEMA 'public';

SELECT * FROM db1_public_sch.table1 ORDER BY c1;

c1 | c2 | c3
----+---+---
1 | 2 | 3
4 | 5 | 6
7 | 8 | 9
(3 rows)
```

To create different views and grant permissions to those views, as user1 on db1, do the following.

```
--As user1 on db1
CREATE VIEW regular_view AS SELECT c1 FROM table1;

GRANT SELECT ON regular_view TO user2;

CREATE MATERIALIZED VIEW mat_view AS SELECT c2 FROM table1;

GRANT SELECT ON mat_view TO user2;

CREATE VIEW late_bind_view AS SELECT c3 FROM public.table1 WITH NO SCHEMA BINDING;

GRANT SELECT ON late_bind_view TO user2;
```

As user2 on db2, run the following cross-database query using the three-part notation to view the particular view.

```
--As user2 on db2
SELECT * FROM db1.public.regular_view;
c1

1
4
7
(3 rows)

SELECT * FROM db1.public.mat_view;
c2

8
5
2
(3 rows)

SELECT * FROM db1.public.late_bind_view;
c3

3
6
9
(3 rows)
```

As user2 on db2, run the following cross-database query using the external schema notation to query the late-binding view.

```
--As user2 on db2
SELECT * FROM db1_public_sch.late_bind_view;
c3
```

```

3
6
9
(3 rows)
```

As user2 on db2, run the following command using connected tables in a single query.

```
--As user2 on db2
CREATE TABLE table1 (a int, b int, c int);

INSERT INTO table1 VALUES (1,2,3), (4,5,6), (7,8,9);

SELECT a AS col_1, (db1.public.table1.c2 + b) AS sum_col2, (db1.public.table1.c3 + c) AS
sum_col3 FROM db1.public.table1, table1 WHERE db1.public.table1.c1 = a;
col_1 | sum_col2 | sum_col3
-----+-----+
1 | 4 | 6
4 | 10 | 12
7 | 16 | 18
(3 rows)
```

The following example lists all databases on the cluster.

```
select database_name, database_owner, database_type
from svv_redshift_databases
where database_name in ('db1', 'db2');

database_name | database_owner | database_type
-----+-----+
db1 | 100 | local
db2 | 100 | local
(2 rows)
```

The following example lists all Amazon Redshift schemas of all databases on the cluster.

```
select database_name, schema_name, schema_owner, schema_type
from svv_redshift_schemas
where database_name in ('db1', 'db2');

database_name | schema_name | schema_owner | schema_type
-----+-----+-----+
db1 | pg_catalog | 1 | local
db1 | public | 1 | local
db1 | information_schema | 1 | local
db2 | pg_catalog | 1 | local
db2 | public | 1 | local
db2 | information_schema | 1 | local
(6 rows)
```

The following example lists all Amazon Redshift tables or views of all databases on the cluster.

```
select database_name, schema_name, table_name, table_type
from svv_redshift_tables
where database_name in ('db1', 'db2') and schema_name in ('public');

database_name | schema_name | table_name | table_type
-----+-----+-----+
db1 | public | late_bind_view | VIEW
db1 | public | mat_view | VIEW
db1 | public | mv_tbl__mat_view__0 | TABLE
```

|          |        |              |       |
|----------|--------|--------------|-------|
| db1      | public | regular_view | VIEW  |
| db1      | public | table1       | TABLE |
| db2      | public | table2       | TABLE |
| (6 rows) |        |              |       |

The following example lists all Amazon Redshift and external schemas of all databases on the cluster.

```
select database_name, schema_name, schema_owner, schema_type
from svv_all_schemas where database_name in ('db1', 'db2') ;

database_name | schema_name | schema_owner | schema_type
-----+-----+-----+-----+
db1 | pg_catalog | 1 | local
db1 | public | 1 | local
db1 | information_schema | 1 | local
db2 | pg_catalog | 1 | local
db2 | public | 1 | local
db2 | information_schema | 1 | local
db2 | db1_public_sch | 1 | external
(7 rows)
```

The following example lists all Amazon Redshift and external tables of all databases on the cluster.

```
select database_name, schema_name, table_name, table_type
from svv_all_tables
where database_name in ('db1', 'db2') and schema_name in ('public');

database_name | schema_name | table_name | table_type
-----+-----+-----+-----+
db1 | public | regular_view | VIEW
db1 | public | mv_tbl_mat_view_0 | TABLE
db1 | public | mat_view | VIEW
db1 | public | late_bind_view | VIEW
db1 | public | table1 | TABLE
db2 | public | table2 | TABLE
(6 rows)
```

## Using cross-database queries with the query editor

You can use cross-database queries to access data from any of the databases on your Amazon Redshift cluster without having to connect to that specific database. When you run cross-database queries on any other unconnected databases, you have read access only to those database objects.

You can query other database objects using fully qualified object names expressed with three-part notation. The full path to any database object consists of three components: database name, schema, and name of the object. An example is `database_name.schema_name.object_name`.

### To use cross-database queries with the query editor

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. Create a cluster to use cross-database queries in the Amazon Redshift query editor. For more information, see [Creating a cluster](#) in the *Amazon Redshift Management Guide*.
3. Enable access to the query editor with the appropriate permissions. For more information, see [Querying a database using the query editor](#) in the *Amazon Redshift Management Guide*.
4. On the navigation menu, choose **Query editor**, then connect to a database in your cluster.

When you connect to the query editor for the first time, Amazon Redshift shows the resources for the connected database by default.

5. Choose the other databases that you have access to view database objects for these other databases. To view objects, make sure that you have the appropriate permissions. After you choose a database, Amazon Redshift shows the list of schemas from the database.

Select a schema to see the list of database objects within that schema.

**Note**

Amazon Redshift doesn't directly support query catalog objects that are part of AWS Glue or federated databases. To query these, first create external schemas that refer to those external data sources in each database.

Amazon Redshift cross-database queries with three-part notation don't support metadata tables under the schemas `information_schema` and `pg_catalog` because these metadata views are specific to a database.

6. (Optional) Filter the list of tables or views for the schema that you selected.

# Sharing data across clusters in Amazon Redshift

With *data sharing*, you can share live data with relative security and ease across Amazon Redshift clusters, AWS accounts, or AWS Regions for read purposes.

Data sharing can improve the agility of your organization. It does this by giving you instant, granular, and high-performance access to data across Amazon Redshift clusters without the need to copy or move it manually. With data sharing, you have live access to data, so that your users can see the most up-to-date and consistent information as it's updated in Amazon Redshift clusters.

For details about SQL commands to create and manage datashares, see the following:

- [CREATE DATASHARE \(p. 754\)](#)
- [ALTER DATASHARE \(p. 629\)](#)
- [DESC DATASHARE \(p. 860\)](#)
- [SHOW DATASHARES \(p. 986\)](#)
- [DROP DATASHARE \(p. 864\)](#)

## Regions where data sharing is available

The following regions have data sharing available:

- US East (Ohio) (us-east-2)
- US East (N. Virginia) (us-east-1)
- US West (N. California) (us-west-1)
- US West (Oregon) (us-west-2)
- Asia Pacific (Hong Kong) (ap-east-1)
- Asia Pacific (Mumbai) (ap-south-1)
- Asia Pacific (Seoul) (ap-northeast-2)
- Asia Pacific (Singapore) (ap-southeast-1)
- Asia Pacific (Sydney) (ap-southeast-2)
- Asia Pacific (Tokyo) (ap-northeast-1)
- China (Beijing) (cn-north-1)
- China (Ningxia) (cn-northwest-1)
- Canada (Central) (ca-central-1)
- Europe (Frankfurt) (eu-central-1)
- Europe (Ireland) (eu-west-1)
- Europe (London) (eu-west-2)
- Europe (Paris) (eu-west-3)
- Europe (Stockholm) (eu-north-1)
- Middle East (Bahrain) (me-south-1)
- South America (São Paulo) (sa-east-1)
- AWS GovCloud (US-East) (us-gov-east-1)

- AWS GovCloud (US-West) (us-gov-west-1)

## Overview of data sharing in Amazon Redshift

With *data sharing*, you can securely and easily share live data across Amazon Redshift clusters for read purposes.

For information about how to get started working with data sharing and manage datashares using the AWS Management Console, see [Getting started data sharing \(p. 333\)](#).

### Topics

- [Data sharing use cases for Amazon Redshift \(p. 318\)](#)
- [Data sharing concepts for Amazon Redshift \(p. 318\)](#)
- [Sharing data at different levels in Amazon Redshift \(p. 320\)](#)
- [Managing data consistency in Amazon Redshift \(p. 320\)](#)
- [Accessing shared data in Amazon Redshift \(p. 320\)](#)
- [Considerations when using data sharing in Amazon Redshift \(p. 320\)](#)
- [How data sharing works in Amazon Redshift \(p. 321\)](#)
- [Controlling access for cross-account datashares \(p. 322\)](#)

## Data sharing use cases for Amazon Redshift

Amazon Redshift data sharing is especially useful for these use cases:

- **Supporting different kinds of business-critical workloads** – Use a central extract, transform, and load (ETL) cluster that shares data with multiple business intelligence (BI) or analytic clusters. This approach provides read workload isolation and chargeback for individual workloads. You can size and scale your individual workload compute according to the workload-specific requirements of price and performance.
- **Enabling cross-group collaboration** – Enable seamless collaboration across teams and business groups for broader analytics, data science, and cross-product impact analysis.
- **Delivering data as a service** – Share data as a service across your organization.
- **Sharing data between environments** – Share data among development, test, and production environments. You can improve team agility by sharing data at different levels of granularity.
- **Licensing access to data in Amazon Redshift** – List Amazon Redshift data sets in the AWS Data Exchange catalog that customers can find, subscribe to, and query in minutes.

## Data sharing concepts for Amazon Redshift

Following are some data sharing terms and concepts:

- A *datashare* is the unit of sharing data in Amazon Redshift. Use datashares to share data in the same AWS account or different AWS accounts. Also, share data for read purposes across different Amazon Redshift clusters.

Each datashare is associated with a specific database in your Amazon Redshift cluster.

A cluster administrator can receive datashares from other clusters, referred to as *inbound* shares. A cluster administrator can also create datashares to share data with other clusters, referred to as *outbound* shares.

- An *AWS Data Exchange datashare* is a unit of licensing for sharing your data through AWS Data Exchange. AWS manages all billing and payments associated with subscriptions to AWS Data Exchange and use of Amazon Redshift data sharing. Approved data providers can add AWS Data Exchange datashares to AWS Data Exchange products. When customers subscribe to a product with AWS Data Exchange datashares, they get access to the datashares in the product.

For more information, see [Working with AWS Data Exchange for Amazon Redshift \(p. 330\)](#).

- *Data producers* (also known as data sharing producers or datashare producers) are clusters that you want to share data from. Producer cluster administrators and database owners can create datashares using the CREATE DATASHARE command. You can add objects such as schemas, tables, views, and SQL user-defined functions (UDFs) from a database that you want the producer cluster to share with consumer clusters for read purposes.

Data producers (also known as providers on AWS Data Exchange) for AWS Data Exchange datashares can license data through AWS Data Exchange. Approved providers can add AWS Data Exchange datashares to AWS Data Exchange products.

When a customer subscribes to a product with AWS Data Exchange datashares, AWS Data Exchange automatically adds the customer as a data consumer on all AWS Data Exchange datashares included with the product. AWS Data Exchange also removes all customers from AWS Data Exchange datashares when their subscription ends. AWS Data Exchange also automatically manages billing, invoicing, payment collection, and payment distribution for paid products with AWS Data Exchange datashares. For more information, see [Working with AWS Data Exchange for Amazon Redshift \(p. 330\)](#). To register as an AWS Data Exchange data provider, see [Getting started as a provider](#).

- *Data consumers* (also known as data sharing consumers or datashare consumers) are clusters that receive datashares from producer clusters.

Amazon Redshift clusters that share data can be in the same or different AWS account or different AWS Regions, so you can share data across organizations and collaborate with other parties. Consumer cluster administrators receive the datashares that they are granted usage for and review the contents of each datashare. To consume shared data, the consumer cluster administrator creates an Amazon Redshift database from the datashare. The administrator then assigns permissions for the database to users and groups in the consumer cluster. After permissions are granted, users and groups can list the shared objects as part of the standard metadata queries, along with the local data on the consumer cluster. They can start querying immediately.

If you are a *consumer with an active AWS Data Exchange subscription* (also known as subscribers on AWS Data Exchange), you can find, subscribe to, and query granular, up-to-date data in Amazon Redshift without the need to extract, transform, and load the data. For more information, see [Working with AWS Data Exchange for Amazon Redshift \(p. 330\)](#).

- *Datashare objects* are objects from specific databases on a cluster that producer cluster administrators can add to datashares to be shared with data consumers. Datashare objects are read-only for data consumers. Examples of datashare objects are tables, views, and user-defined functions. You can add datashare objects to datashares while creating datashares or editing a datashare at any time.
- *Provisioned cluster namespaces* are identifiers that identify Amazon Redshift clusters. A namespace globally unique identifier (GUID) is automatically created during Amazon Redshift cluster creation and attached to the cluster. A namespace Amazon Resource Name (ARN) is in the `arn: {partition}:redshift:{region}:{account-id}:namespace:{namespace-guid}` format. You can see the namespace of an Amazon Redshift cluster on the cluster details page on the Amazon Redshift console.

In the data sharing workflow, the namespace GUID value and the cluster namespace ARN are used to share data with clusters in the AWS account. You can also find the namespace for the current cluster by using the `current_namespace` function.

- *Serverless namespaces* are identifiers that identify Amazon Redshift Serverless. A namespace globally unique identifier (GUID) is automatically created during Amazon Redshift Serverless creation and attached to the instance. A serverless namespace Amazon Resource Name (ARN) is in the `arn:`

{partition}:redshift-serverless:{region}:{account-id}:namespace/{namespace-guid} format.

- AWS accounts can be consumers for datashares and are each represented by a 12-digit AWS account ID.

## Sharing data at different levels in Amazon Redshift

With Amazon Redshift, you can share data at different levels. These levels include databases, schemas, tables, views (including regular, late-binding, and materialized views), and SQL user-defined functions (UDFs). You can create multiple datashares for a given database. A datashare can contain objects from multiple schemas in the database on which sharing is created.

By having this flexibility in sharing data, you get fine-grained access control. You can tailor this control for different users and businesses that need access to Amazon Redshift data.

## Managing data consistency in Amazon Redshift

Amazon Redshift provides transactional consistency on all producer and consumer clusters and shares up-to-date and consistent views of the data with all consumers.

You can continuously update data on the producer cluster. All queries on a consumer cluster within a transaction read the same state of the shared data. Amazon Redshift doesn't consider the data that was changed by another transaction on the producer cluster that was committed after the beginning of the transaction on the consumer cluster. After the data change is committed on the producer cluster, new transactions on the consumer cluster can immediately query the updated data.

The strong consistency removes the risks of lower-fidelity business reports that might contain invalid results during sharing of data. This factor is especially important for financial analysis or where the results might be used to prepare datasets that are used to train machine learning models.

## Accessing shared data in Amazon Redshift

You can discover shared data using standard SQL interfaces, JDBC or ODBC drivers, and the Data API. You can also query data with high performance from familiar business intelligence (BI) and analytic tools. You can perform queries by referring to the objects from other Amazon Redshift databases that are both local to and remote from your cluster that you have permissions to access.

You can do so simply by staying connected to local databases in your cluster. Then you can create consumer databases from datashares to consume shared data.

After you have done so, you can perform cross-database queries joining the datasets. You can query objects in consumer databases using the 3-part notation (*consumer\_database\_name.schema\_name.table\_name*). You can also query using external schema links to schemas in the consumer database. You can query both local data and data shared from other clusters within the same query. Such a query can reference objects from the current connected database and from other nonconnected databases, including consumer databases created from datashares.

## Considerations when using data sharing in Amazon Redshift

Following are considerations for working with Amazon Redshift data sharing. For information on data sharing limitations, see [Limitations for data sharing \(p. 321\)](#).

- Amazon Redshift data sharing is supported for all provisioned ra3 cluster types (ra3.16xlarge, ra3.4xlarge, and ra3.xlplus) and Amazon Redshift Serverless.
- You can share data across provisioned clusters, serverless workgroups, Availability Zones, AWS accounts, and AWS Regions. You can share between cluster types as well as between provisioned clusters and serverless.
- Cross-Region data sharing comes with additional cross-region data transfer charges.
- You can use AWS Data Exchange datashares to license query access to your data via AWS Data Exchange.
- You only need datashares when you are sharing data between different Amazon Redshift provisioned clusters or serverless workgroups. Within the same cluster, you can query another database using simple three-part notation database.schema.table as long as you have the required privileges on the objects in the other database.
- As a datashare user, you continue to connect to your local cluster database only. You can't connect to the databases created from a datashare but can read from those databases.
- The consumer is charged for all compute and cross-region data transfer fees required to query the producer's data. The producer is charged for the underlying storage of data in their provisioned cluster or serverless namespace.
- The performance of the queries on shared data depends on the compute capacity of the consumer clusters.

## Limitations for data sharing

The following are limitations when working with datashares in Amazon Redshift:

- For cross-account and cross-Region data sharing, both the producer and consumer clusters and serverless namespaces must be encrypted. This is for security purposes. However, they don't need to share the same encryption key.
- You can only share SQL UDFs through datashares. Python and lambda UDFs aren't supported.
- If the producer database has specific collation, use the same collation settings for the consumer database.
- Amazon Redshift doesn't support adding external schemas, tables, or late-binding views on external tables to datashares.
- Amazon Redshift doesn't support nested SQL user-defined functions on producer clusters.
- Amazon Redshift doesn't support sharing tables with interleaved sort keys and views that refer to tables with interleaved sort keys.
- Consumers can't add datashare objects to another datashare. Additionally, consumers can't add views referencing datashare objects to another datashare.

## How data sharing works in Amazon Redshift

Data sharing between Amazon Redshift clusters is a two-step process:

1. The producer cluster administrator who wants to share data creates an Amazon Redshift datashare. The producer cluster administrator then adds the needed database objects. These might be schemas, tables, and views to the datashare and specifies a list of consumers that the objects to be shared with.

Use the following system views to see datashare objects and data consumer information for outbound datashares:

- [SVV\\_DATASHARES \(p. 1422\)](#)
- [SVV\\_DATASHARE\\_CONSUMERS \(p. 1423\)](#)
- [SVV\\_DATASHARE\\_OBJECTS \(p. 1424\)](#)

2. The consumer cluster administrators look at the datashares for which they're granted use and review the contents of each datashare by viewing inbound datashares using [SVV\\_DATASHARES \(p. 1422\)](#).

To consume shared data, each consumer cluster administrator creates an Amazon Redshift database from the datashare. The administrator then assigns permissions to appropriate users and groups in the consumer cluster. Users and groups can list the shared objects as part of the standard metadata queries by viewing the following metadata system views and can start querying data immediately.

- [SVV\\_REDSHIFT\\_COLUMNS \(p. 1444\)](#)
- [SVV\\_REDSHIFT\\_DATABASES \(p. 1445\)](#)
- [SVV\\_REDSHIFT\\_FUNCTIONS \(p. 1446\)](#)
- [SVV\\_REDSHIFT\\_SCHEMAS \(p. 1447\)](#)
- [SVV\\_REDSHIFT\\_TABLES \(p. 1448\)](#)

To view objects of both Amazon Redshift local and shared schemas and external schemas, use the following metadata system views to query them.

- [SVV\\_ALL\\_COLUMNS \(p. 1412\)](#)
- [SVV\\_ALL\\_SCHEMAS \(p. 1413\)](#)
- [SVV\\_ALL\\_TABLES \(p. 1414\)](#)

To learn how Amazon Redshift data sharing works, watch the following video: [Amazon Redshift data sharing workflow](#).

To learn about Amazon Redshift data sharing use cases, watch the following video: [Amazon Redshift data sharing use cases](#).

## Controlling access for cross-account datashares

Following are considerations for controlling cross-account datashare access.

### Topics

- [Managing datashares at different states \(p. 322\)](#)
- [Managing access to data sharing API operations with IAM policies \(p. 323\)](#)
- [Managing cluster encryption \(p. 324\)](#)
- [Integrating Amazon Redshift data sharing with AWS CloudTrail \(p. 324\)](#)

## Managing datashares at different states

With cross-account datashares, there are different statuses of datashares that require your actions. Your datashare can have the status are active, action required, or inactive.

Following describes each datashare status and its required action:

- When a producer cluster administrator creates a datashare, the datashare status on the producer cluster is **Pending authorization**. The producer cluster administrator can authorize data consumers to access the datashare. There isn't any action for the consumer cluster administrator.
- When a producer cluster administrator authorizes the datashare, the datashare status becomes **Authorized** on the producer cluster. There isn't any action for the producer cluster administrator. When there is at least one association with a data consumer for the datashare, the datashare status changes from **Authorized** to **Active**.

The datashare share status then becomes **Available (Action required on the Amazon Redshift console)** on the consumer cluster. The consumer cluster administrator can associate the datashare with

data consumers or reject the datashare. The consumer cluster administrator can also use the AWS CLI command `describeDatashareForConsumer` to view the status of datashares. Or the administrator can use the CLI command `describeDatashare` and provide the datashare Amazon Resource Name (ARN) to view the status of the datashare.

- When the consumer cluster administrator associates a datashare with data consumers, the datashare status becomes **Active** on the producer cluster. When there is at least one association with a data consumer for the datashare, the datashare status changes from **Authorized** to **Active**. There isn't any action required for the producer cluster administrator.

The datashare status becomes **Active** on the consumer cluster. There isn't any action required for the consumer cluster administrator.

- When the consumer cluster administrator removes a consumer association from a datashare, the datashare status becomes either **Active** or **Authorized**. It becomes **Active** when there is at least one association exists for the datashare with another data consumer. It becomes **Authorized** when there isn't any consumer association with the datashare on the producer cluster. There isn't any action for the producer cluster administrator.

The datashare status becomes **Action required** on the consumer cluster if all associations are removed. The consumer cluster administrator can reassociate a datashare with data consumers when the datashare is available to the consumers.

- When a consumer cluster administrator declines a datashare, the datashare status on the producer cluster becomes **Action required** and **Declined** on the consumer cluster. The producer cluster administrator can reauthorize the datashare. There isn't any action for the consumer cluster administrator.
- When the producer cluster administrator removes authorization from a datashare, the datashare's status becomes **Action required** on the producer cluster. The producer cluster administrator can choose to reauthorize the datashare, if necessary. There isn't any action required for the consumer cluster administrator.

## Managing access to data sharing API operations with IAM policies

To control the access to the data sharing API operations, use IAM action-based policies. For information about how to manage IAM policies, see [Managing IAM policies in the \*IAM User Guide\*](#).

For information on the permissions required to use the data sharing API operations, see [Permissions required to use the data sharing API operations](#) in the [Amazon Redshift Management Guide](#).

To make cross-account data sharing more secure, you can use a conditional key `ConsumerIdentifier` for the `AuthorizeDataShare` and `DeauthorizeDataShare` API operations. By doing this, you can explicitly control which AWS accounts can make calls to the two API operations.

You can deny authorizing or deauthorizing data sharing for any consumer that isn't your own account. To do so, specify the AWS account number in the IAM policy.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "VisualEditor0",
 "Effect": "Deny",
 "Action": [
 "redshift:AuthorizeDataShare",
 "redshift:DeauthorizeDataShare"
],
 "Resource": "*"
 }
]}
```

```
 "Condition": {
 "StringNotEquals": {
 "redshift:ConsumerIdentifier": "555555555555"
 }
 }
]
}
```

You can allow a producer with a DataShareArn **testshare2** to explicitly share with a consumer with an AWS account of 111122223333 in the IAM policy.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "VisualEditor0",
 "Effect": "Allow",
 "Action": [
 "redshift:AuthorizeDataShare",
 "redshift:DeauthorizeDataShare"
],
 "Resource": "arn:aws:redshift:us-
east-1:666666666666:datashare:af06285e-8a45-4ee9-b598-648c218c8ff1/testshare2",
 "Condition": {
 "StringEquals": {
 "redshift:ConsumerIdentifier": "111122223333"
 }
 }
]
]
}
```

## Managing cluster encryption

To share data across AWS account, both the producer and consumer clusters must be encrypted.

In Amazon Redshift, you can turn on database encryption for your clusters to help protect data at rest. When you turn on encryption for a cluster, the data blocks and system metadata are encrypted for the cluster and its snapshots. You can turn on encryption when you launch your cluster, or you can modify an unencrypted cluster to use AWS Key Management Service (AWS KMS) encryption. For more information about Amazon Redshift database encryption, see [Amazon Redshift database encryption](#) in the *Amazon Redshift Management Guide*.

To protect data in transit, all data is encrypted in transit through the encryption schema of the producer cluster. The consumer cluster adopts this encryption schema when data is loaded. The consumer cluster then operates as a normal encrypted cluster. Communications between the producer and consumer are also encrypted using a shared key schema. For more information about encryption in transit, [Encryption in transit](#).

## Integrating Amazon Redshift data sharing with AWS CloudTrail

Data sharing is integrated with AWS CloudTrail. CloudTrail is a service that provides a record of actions taken by a user, a role, or an AWS service in Amazon Redshift. CloudTrail captures all API calls for data sharing as events. The calls captured include calls from the AWS CloudTrail console and code calls to the data sharing operations. For more information about Amazon Redshift integration with AWS CloudTrail, see [Logging with CloudTrail](#).

For more information about CloudTrail, see [How CloudTrail works](#).

# Working with views in Amazon Redshift data sharing

A producer cluster can share regular, late-binding, and materialized views. When sharing regular or late-binding views, you don't have to share the base tables. The following table shows how views are supported with data sharing.

| View name         | Can this view be added to a datashare? | Can a consumer create this view on datashare objects across clusters? |  |
|-------------------|----------------------------------------|-----------------------------------------------------------------------|--|
| Regular view      | Yes                                    | No                                                                    |  |
| Late-binding view | Yes                                    | Yes                                                                   |  |
| Materialized view | Yes                                    | Yes, but only with a complete refresh                                 |  |

The following query shows the output of a regular view that is supported with data sharing. For information about regular view definition, see [CREATE VIEW \(p. 851\)](#).

```
SELECT * FROM ticket_db.public.myevent_regular_vw
ORDER BY eventid LIMIT 5;

eventid | eventname
-----+-----
 3835 | LeAnn Rimes
 3967 | LeAnn Rimes
 4856 | LeAnn Rimes
 4948 | LeAnn Rimes
 5131 | LeAnn Rimes
```

The following query shows the output of a late-binding view that is supported with data sharing. For information about late-binding view definition, see [CREATE VIEW \(p. 851\)](#).

```
SELECT * FROM ticket_db.public.event_lbv
ORDER BY eventid LIMIT 5;

eventid | venueid | catid | dateid | eventname | starttime
-----+-----+-----+-----+-----+-----+
 1 | 305 | 8 | 1851 | Gotterdamerung | 2008-01-25 14:30:00
 2 | 306 | 8 | 2114 | Boris Godunov | 2008-10-15 20:00:00
 3 | 302 | 8 | 1935 | Salome | 2008-04-19 14:30:00
 4 | 309 | 8 | 2090 | La Cenerentola (Cinderella) | 2008-09-21 14:30:00
 5 | 302 | 8 | 1982 | Il Trovatore | 2008-06-05 19:00:00
```

The following query shows the output of a materialized view that is supported with data sharing. For information about materialized view definition, see [CREATE MATERIALIZED VIEW \(p. 796\)](#).

```
SELECT * FROM ticket_db.public.tickets_mv;

catgroup | qtysold
-----+-----
Concerts | 195444
Shows | 149905
```

You can maintain common tables across all tenants in a producer cluster. You can also share subsets of data filtered by dimension columns, such as `tenant_id` (`account_id` or `namespace_id`), to consumer clusters. To do this, you can define a view on the base table with a filter on these ID columns, for example `current_aws_account = tenant_id`. On the consumer side, when you query the view, you see only the rows that qualify for your account. To do this, you can use the Amazon Redshift context functions `current_aws_account` and `current_namespace`.

The following query returns the account ID in which the current Amazon Redshift cluster resides. You can run this query if you are connected to Amazon Redshift.

```
select current_user, current_aws_account;

current_user | current_aws_account
-----+-----
dwuser | 111111111111
(1row)
```

The following query returns the namespace of the current Amazon Redshift cluster. You can run this query if you are connected to the database.

```
select current_user, current_namespace;

current_user | current_namespace
-----+-----
dwuser | 86b5169f-01dc-4a6f-9fbb-e2e24359e9a8
(1 row)
```

## Managing the data sharing lifecycle in Amazon Redshift

The lifecycle of a datashare object follows the same paradigm as for any Amazon Redshift object, from creation to deletion.

As part of this lifecycle, Amazon Redshift applies drop user checks for datashare objects. You can't drop a user if the user owns any datashare objects or if the user has any permissions on datashare objects. For more information, see [DROP USER \(p. 876\)](#).

If you attempt to drop such a user, you receive one of the following errors.

```
ERROR: user "username" can't be dropped because the user owns some object [SQL State =55006] ERROR: user "username" can't be dropped because the user has a privilege on some object [SQL State =55006]
```

## Managing permissions for datashares in Amazon Redshift

As a producer cluster administrator, you retain control for the datasets you are sharing. You can add new objects to or remove them from the datashare. You can also grant or revoke access to datashares as a whole for the consumer clusters, AWS accounts, or AWS Regions. When permissions are revoked,

consumer clusters immediately lose access to the shared objects and stop seeing them in the list of INBOUND datashares in SVV\_DATASHARES.

The following example creates a datashare salesshare and adds a table public.ticket\_sales\_redshift to salesshare. It also grants usage permissions on salesshare to a cluster namespace.

```
CREATE DATASHARE salesshare;

ALTER DATASHARE salesshare ADD TABLE public.ticket_sales_redshift;

GRANT USAGE ON DATASHARE salesshare TO NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

For CREATE DATASHARE, superusers and database owners can create datashares. For more information, see [CREATE DATASHARE \(p. 754\)](#). For ALTER DATASHARE, the owner of the datashare with the required permissions on the datashare objects to be added or removed can alter the datashare. For information, see [ALTER DATASHARE \(p. 629\)](#).

As a producer administrator, when you drop a datashare, it stops being listed on consumer clusters. The databases and schema references created on the consumer cluster from the dropped datashare continue to exist with no objects in them. The consumer cluster administrator must delete these databases manually.

On the consumer side, a consumer cluster administrator can determine which users and groups should get access to the shared data. An administrator can control access at the database or schema level. To control access at the schema level, the administrator must create an external schema from the Amazon Redshift database created from the datashare.

The following example grants permissions to access a shared table at the database level and schema level.

```
GRANT USAGE ON DATABASE sales_db TO Bob;

CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE sales_db;

GRANT USAGE ON SCHEMA sales_schema TO GROUP Analyst_group;
```

To further restrict access, you can create views on top of shared objects, exposing only the necessary data. You can then use these views to give access to the users and groups.

Once the users are granted access to the database or schema, they will have access to all shared objects in that database or schema.

## Tracking usage and auditing in data sharing in Amazon Redshift

By auditing data sharing, producers can track the datashare evolution. For example, auditing helps track when datashares are created, objects are added or removed, and permissions are granted or revoked to Amazon Redshift clusters, AWS accounts, or AWS Regions.

In addition to auditing, producers and consumers track datashare usage at various granularities, such as account, cluster, and object levels. For more information about tracking usage and auditing views, see [SVL\\_DATASHARE\\_CHANGE\\_LOG \(p. 1631\)](#) and [SVL\\_DATASHARE\\_USAGE\\_PRODUCER \(p. 1634\)](#).

# Cluster management and data sharing in Amazon Redshift

Consider the following for cluster management tasks:

- Data sharing continues to work when clusters are resized.
- Data sharing continues to work when the producer cluster is paused.
- Data sharing works across Availability Zones.
- For standard datashares, consider the following:
  - When a producer cluster is deleted, Amazon Redshift deletes the datashares created by the producer cluster. When a producer cluster is backed up and restored, the created datashares still persist on the restored cluster. However, datashare permissions granted to other clusters are no longer valid on the restored cluster. Re-grant usage permissions of datashares to desired consumer clusters. The consumer database on the consumer cluster points to the datashare from the original cluster where the snapshot is taken. To query the shared data from the restored cluster, the consumer cluster administrator creates a different database. Or the administrator can drop and recreate an existing consumer database to use the datashare from the newly restored cluster.
  - When a consumer cluster is deleted and restored from a snapshot, the previous access shared to this cluster would no longer be valid and visible. If access to datashares is still required on the restored consumer cluster, the producer cluster administrator must grant usage of datashares to the restored consumer cluster again. The consumer cluster administrator must drop any stale consumer databases created from the inactive datashares. Then the administrator must recreate the consumer database from the datashare, after the producer re-granted the permissions. As the cluster namespace GUID is different on a restored cluster from the original cluster, re-grant datashare permissions when the consumer or producer cluster is restored from backup.
- For AWS Data Exchange datashares, consider the following:
  - When a producer cluster is deleted, Amazon Redshift deletes the datashares created by the producer cluster. When a producer cluster is backed up and restored, the created datashares still persist on the restored cluster. For data subscribers to be able to continue accessing the data, create the AWS Data Exchange datashares again and publish them to the product's data sets. The consumer database on the consumer cluster points to the datashare from the original cluster where the snapshot is taken. To query the shared data from the restored cluster, the consumer cluster administrator creates a different database, or drops and recreates an existing consumer database to use the newly created AWS Data Exchange datashare from the newly restored cluster.
  - When a consumer cluster is deleted and restored from a snapshot, the previous access shared to this cluster remains valid and visible. Consumer cluster administrator must drop any stale consumer databases created from the inactive datashares and recreate the consumer database from the datashare after the producer regrants the permissions. As the cluster namespace GUID is different on a restored cluster from the original cluster, regrant datashare permissions when the producer cluster is restored from backup.
  - We recommend that you don't delete your cluster if you have any AWS Data Exchange datashares. Performing this type of alteration can breach data product terms in AWS Data Exchange.

## Integrating Amazon Redshift data sharing with business intelligence tools

To integrate data sharing with business intelligence (BI) tools, we recommend that you use the Amazon Redshift JDBC or ODBC drivers. You can also query the following system metadata views to view information about Amazon Redshift database objects.

- These views include objects from the Amazon Redshift database that you are currently connected to. They also include objects from all the other databases that you have access to on the cluster, including the databases created from the datashares.
  - [SVV\\_REDSHIFT\\_COLUMNS \(p. 1444\)](#)
  - [SVV\\_REDSHIFT\\_DATABASES \(p. 1445\)](#)
  - [SVV\\_REDSHIFT\\_FUNCTIONS \(p. 1446\)](#)
  - [SVV\\_REDSHIFT\\_SCHEMAS \(p. 1447\)](#)
  - [SVV\\_REDSHIFT\\_TABLES \(p. 1448\)](#)
- In addition, these views include external objects that you can query using Amazon Redshift Spectrum and federated queries as part of their respective databases. To see the full list of local cluster and consumer databases that Amazon Redshift uses to navigate these objects, use the SVV\_REDSHIFT\_DATABASES view.
  - [SVV\\_ALL\\_COLUMNS \(p. 1412\)](#)
  - [SVV\\_ALL\\_SCHEMAS \(p. 1413\)](#)
  - [SVV\\_ALL\\_TABLES \(p. 1414\)](#)
- You can also use the SVV\_DATASHARE\* system views to view metadata for datashares on any cluster. If you have explicitly created EXTERNAL SCHEMA references to schemas in other databases, Amazon Redshift also returns the objects in SVV\_EXTERNAL\* system views.
  - [SVV\\_DATASHARES \(p. 1422\)](#)
  - [SVV\\_DATASHARE\\_CONSUMERS \(p. 1423\)](#)
  - [SVV\\_DATASHARE\\_OBJECTS \(p. 1424\)](#)

Amazon Redshift JDBC and ODBC drivers support the GetCatalogs API operation in the drivers, which returns the list of all databases including those created from datashares. The drivers also support downstream operations, such as GetSchemas, GetTables, and so on, that return data from all the databases that GetCatalogs returns. The drivers provide this support even when the catalog isn't explicitly specified in the call. For more information about JDBC or ODBC drivers, see [Configuring connections in Amazon Redshift](#) in the *Amazon Redshift Management Guide*.

You can't connect to consumer databases created from datashares directly. Connect to local databases on your cluster. If you have a connection switching user interface in your tool, the list of databases should include only the local cluster databases. The list should exclude consumer databases created from datashares to provide the best experience. You can use an option in the SVV\_REDSHIFT\_DATABASES view to filter databases.

## Accessing metadata for datashares in Amazon Redshift

To help cluster administrators discover datashares, Amazon Redshift provides a set of metadata views to list datashares. These views list datashares created in your cluster and also those received from other clusters within the same account, from other accounts, or other AWS Regions. These views display the following information:

- Datashares that are shared and received by the clusters
- Contents of database objects in the datashares, including the basic share metadata, objects, and consumers

Use SVV\_DATASHARES to view a list of all datashares created in your cluster (outbound) and shared from others (inbound). For more information, see [SVV\\_DATASHARES \(p. 1422\)](#).

Use SVV\_DATASHARE\_CONSUMERS to view a list of data consumers. For more information, see [SVV\\_DATASHARE\\_CONSUMERS \(p. 1423\)](#).

Use SVV\_DATASHARE\_OBJECTS to view a list of objects in all datashares created in your cluster (outbound) and shared from others (inbound). For more information, see [SVV\\_DATASHARE\\_OBJECTS \(p. 1424\)](#).

## Working with AWS Data Exchange for Amazon Redshift

*AWS Data Exchange for Amazon Redshift* makes it convenient to license access to your Amazon Redshift data through AWS Data Exchange. When a customer subscribes to a product with AWS Data Exchange datashares, AWS Data Exchange automatically adds the customer as a data consumer on all AWS Data Exchange datashares included with the product. Invoices are automatically generated, and payments are centrally collected and automatically disbursed through AWS Marketplace Entitlement Service.

Providers can license data in Amazon Redshift at a granular level, such as schemas, tables, views, and user-defined functions. You can use the same AWS Data Exchange datashare across multiple AWS Data Exchange products. Any objects added to the AWS Data Exchange datashare is available to consumers. Producers can view all AWS Data Exchange datashares managed by AWS Data Exchange on their behalf using Amazon Redshift API operations, SQL commands, and the Amazon Redshift console. Customers who subscribe to a product AWS Data Exchange datashares have read-only access to the objects in the datashares.

Customers who want to consume third-party producer data can browse the AWS Data Exchange catalog to discover and subscribe to datasets in Amazon Redshift. After their AWS Data Exchange subscription is active, they can create a database from the datashare in their cluster and query the data in Amazon Redshift.

## How AWS Data Exchange datashares work

### Managing AWS Data Exchange datashares as a producer administrator

If you are a data producer (also known as a provider on AWS Data Exchange), you can create AWS Data Exchange datashares that connect to your Amazon Redshift databases. To add AWS Data Exchange datashares to products on AWS Data Exchange, you must be a registered AWS Data Exchange provider.

For more information on how to get started with AWS Data Exchange datashares, see [Sharing licensed Amazon Redshift data on AWS Data Exchange \(p. 346\)](#).

### Using AWS Data Exchange datashares as a consumer with an active AWS Data Exchange subscription

If you are a consumer with an active AWS Data Exchange subscription (also known as a subscriber on AWS Data Exchange), you can browse the AWS Data Exchange catalog on the AWS Data Exchange console to discover products containing AWS Data Exchange datashares.

After you subscribe to a product that contains AWS Data Exchange datashares, create a database from the datashare within your cluster. You can then query the data in Amazon Redshift directly without extracting, transforming, and loading the data.

For more information on how to get started with AWS Data Exchange datashares, see [Sharing licensed Amazon Redshift data on AWS Data Exchange \(p. 346\)](#).

## Considerations when using AWS Data Exchange for Amazon Redshift

When using AWS Data Exchange for Amazon Redshift, consider the following:

- Both producers and consumers must use the RA3 instance types to use Amazon Redshift datashares. Producers must use the RA3 instance types with the latest Amazon Redshift cluster version.
- Both the producer and consumer clusters must be encrypted.
- You must be registered as an AWS Data Exchange provider to list products on AWS Data Exchange, including products that contain AWS Data Exchange datashares. For more information, see [Getting started as a provider](#).
- You don't need to be a registered AWS Data Exchange provider to find, subscribe to, and query Amazon Redshift data through AWS Data Exchange.
- To control access to your data, create AWS Data Exchange datashares with the publicly accessible setting turned on. To alter an AWS Data Exchange datashare to turn off the publicly accessible setting, set the session variable to allow ALTER DATASHARE SET PUBLICACCESSIBLE FALSE. For more information, see [ALTER DATASHARE usage notes \(p. 630\)](#).
- Producers can't manually add or remove consumers from AWS Data Exchange datashares because access to the datashares is granted based on having an active subscription to an AWS Data Exchange product that contains the AWS Data Exchange datashare.
- Producers can't view the SQL queries that consumers run. They can only view metadata, such as the number of queries or the objects consumers query, through Amazon Redshift tables that only the producer can access. For more information, see [Tracking usage and auditing in data sharing in Amazon Redshift \(p. 327\)](#).
- We recommend that you make your datashares publicly accessible. If you don't, subscribers on AWS Data Exchange with publicly accessible consumer clusters won't be able to use your datashare.
- We recommend that you don't delete an AWS Data Exchange datashare shared to other AWS accounts using the DROP DATASHARE statement. If you do, the AWS accounts that have access to the datashare will lose access. This action is irreversible. Performing this type of alteration can breach data product terms in AWS Data Exchange. If you want to delete an AWS Data Exchange datashare, see [DROP DATASHARE usage notes \(p. 864\)](#).
- For cross-Region data sharing, you can create AWS Data Exchange datashares to share licensed data.

## AWS Lake Formation-managed Redshift datashares

Using AWS Lake Formation, you can centrally define and enforce database, table, column, and row-level access permissions of Amazon Redshift datashares and restrict user access to objects within a datashare. By sharing data through Lake Formation, you can define permissions in Lake Formation and apply those permissions to any datashare and its objects. For example, if you have a table containing employee information, you can use Lake Formation's column-level filters to prevent employees who don't work in the HR department from seeing personally identifiable information (PII), such as a social security number. For more information about data filters, see [Data filtering and cell-level security in Lake Formation](#) in the *AWS Lake Formation Developer Guide*.

You can also use tags in Lake Formation to configure permissions on Lake Formation resources. For more information, see [Lake Formation Tag-based access control](#).

Amazon Redshift currently supports data sharing via Lake Formation when sharing within the same account or across accounts. Cross-Region sharing is currently not supported.

The following is a high-level overview of how to use Lake Formation to control datashare permissions:

1. In Amazon Redshift, the producer account administrator creates a datashare on the producer cluster and grants usage to a Lake Formation account.
2. The producer cluster administrator authorizes the Lake Formation account to access the datashare.
3. The Lake Formation administrator discovers and registers the datashares. They must also discover the AWS Glue ARNs they have access to and associate the datashares with an AWS Glue Data Catalog ARN. If you're using the AWS CLI you can discover and accept datashares with the Redshift CLI operations `describe-data-shares` and `associate-data-share-consumer`. To register a datashare, use the Lake Formation CLI operation `register-resource`.
4. The Lake Formation administrator creates a federated database in the AWS Glue Data Catalog, and configures Lake Formation permissions to control user access to objects within the datashare. For more information about federated databases in AWS Glue, see [Managing permissions for data in an Amazon Redshift datashare](#).
5. The Lake Formation administrator discovers the AWS Glue databases they have access to and associates the datashare with an AWS Glue Data Catalog ARN.
6. The Redshift administrator discovers the AWS Glue database ARNs they have access to, creates an external database in the Amazon Redshift consumer cluster using a AWS Glue database ARN, and grants usage to [database users authenticated with IAM credentials](#) to start querying the Amazon Redshift database.
7. Database users can use the views `SVV_EXTERNAL_TABLES` and `SVV_EXTERNAL_COLUMNS` to find all of the tables or columns within the AWS Glue database that they have access to, and then they can query the AWS Glue database's tables.
8. When the producer cluster administrator decides to no longer share the data with the consumer cluster, the producer cluster administrator can revoke usage, deauthorize, or delete the datashare from Redshift. The associated permissions and objects in Lake Formation are not automatically deleted.

For more information about sharing a datashare with AWS Lake Formation as a producer cluster administrator, see [Working with Lake Formation-managed datashares as a producer \(p. 353\)](#). To consume the shared data from the producer cluster, see [Working with Lake Formation-managed datashares as a consumer \(p. 354\)](#).

## Considerations and limitations when using AWS Lake Formation with Amazon Redshift

The following are considerations and limitations for sharing Amazon Redshift data via Lake Formation. For information on data sharing considerations and limitations, see [Considerations when using data sharing in Amazon Redshift](#). For information about Lake Formation limitations, see [Notes on working with Amazon Redshift datashares in Lake Formation](#).

- Sharing a datashare to Lake Formation across Regions is currently unsupported.
- If column-level filters are defined for a user on a shared relation, performing a `SELECT *` operation returns only the columns the user has access to.
- Cell-level filters from Lake Formation are unsupported.
- If you created and shared a view and its tables to Lake Formation, you can configure filters to manage access of the tables, Amazon Redshift enforces Lake Formation defined policies when consumer cluster users access shared objects. When a user accesses a view shared with Lake Formation, Redshift enforces only the Lake Formation policies defined on the view and not the tables contained within the view. However, when users directly access the table, Redshift enforces the defined Lake Formation policies on the table.
- You can't create materialized views based on a table if the table has Lake Formation filters configured.

- The Lake Formation administrator must have [data lake administrator](#) permissions and the [required permissions to accept a datashare](#).
- The producer consumer cluster must be an RA3 cluster with the latest Amazon Redshift cluster version or a serverless workgroup to share datashares via Lake Formation.
- Both the producer and consumer clusters must be encrypted.
- Redshift row-level and column-level access control policies implemented in the producer cluster are ignored when the datashare is shared to Lake Formation. The Lake Formation administrator must configure these policies in Lake Formation. The producer cluster administrator can turn off RLS for a table by using the [ALTER TABLE](#) command.
- Sharing datashares via Lake Formation is only available to users who have access to both Redshift and Lake Formation.

## Getting started data sharing

You can get started with data sharing by using either the SQL interface or the Amazon Redshift console.

### Topics

- [Getting started data sharing using the SQL interface \(p. 333\)](#)
- [Getting started data sharing using the console \(p. 356\)](#)
- [Getting started data sharing with AWS CloudFormation \(p. 365\)](#)

## Getting started data sharing using the SQL interface

You can share data for read purposes across different Amazon Redshift clusters within or across AWS accounts, or across AWS Regions.

### Topics

- [Sharing data within an AWS account \(p. 333\)](#)
- [Sharing data across AWS accounts \(p. 339\)](#)
- [Sharing data across AWS Regions \(p. 344\)](#)
- [Sharing licensed Amazon Redshift data on AWS Data Exchange \(p. 346\)](#)
- [Working with AWS Lake Formation-managed datashares \(p. 353\)](#)

## Sharing data within an AWS account

You can share data for read purposes across different Amazon Redshift clusters within an AWS account.

### To share data for read purposes as a producer cluster administrator or database owner

1. Create datashares in your cluster. For more information, see [CREATE DATASHARE \(p. 754\)](#).

```
CREATE DATASHARE salesshare;
```

Cluster superuser and database owners can create datashares. Each datashare is associated with a database during creation. Only objects from that database can be shared in that datashare. Multiple datashares can be created on the same database with the same or different granularity of objects. There is no limit on the number of datashares a cluster can create.

You can also use the Amazon Redshift console to create datashares. For more information, see [Creating datashares \(p. 357\)](#).

2. Delegate permissions to operate on the datashare. For more information, see [GRANT \(p. 887\)](#) or [REVOKE \(p. 916\)](#).

The following example grants permissions to dbuser on salesshare.

```
GRANT ALTER, SHARE ON DATASHARE salesshare TO dbuser;
```

Cluster superusers and the owners of the datashare can grant or revoke modification permissions on the datashare to additional users.

3. Add objects to or remove objects from datashares. To add objects to a datashare, add the schema before adding objects. When you add a schema, Amazon Redshift doesn't add all the objects under it. Make sure to add these explicitly. For more information, see [ALTER DATASHARE \(p. 629\)](#).

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare ADD TABLE public.ticket_sales_redshift;
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

You can also add views to a datashare.

```
CREATE VIEW public.sales_data_summary_view AS SELECT * FROM
public.ticket_sales_redshift;
ALTER DATASHARE salesshare ADD TABLE public.sales_data_summary_view;
```

Use `ALTER DATASHARE` to share schemas, and tables, views, and functions in a given schema. Superusers, datashare owners, or users who have `ALTER` or `ALL` permission on the datashare can alter the datashare to add objects to or remove objects from it. Users should have the permissions to add or remove objects from the datashare. Users should also be the owners of the objects or have `SELECT`, `USAGE`, or `ALL` permissions on the objects.

Use the `INCLUDENEW` clause to add any new tables, views, or SQL user-defined functions (UDFs) created in a specified schema to the datashare. Only superusers can change this property for each datashare-schema pair.

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare SET INCLUDENEW = TRUE FOR SCHEMA PUBLIC;
```

You can also use the Amazon Redshift console to add or remove objects from datashares. For more information, see [Adding datashare objects to datashares \(p. 358\)](#), [Removing datashare objects from datashares \(p. 361\)](#), and [Editing datashares created in your account \(p. 362\)](#).

4. Add consumers to or remove consumers from datashares. The following example adds the consumer cluster namespace to salesshare. The namespace is the namespace globally unique identifier (GUID) of the consumer cluster in the account. For more information, see [GRANT \(p. 887\)](#) or [REVOKE \(p. 916\)](#).

```
GRANT USAGE ON DATASHARE salesshare TO NAMESPACE
'13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

You can only grant permissions to one datashare consumer in a `GRANT` statement.

Cluster superusers and the owners of datashare objects or users that have `SHARE` privilege on the datashare can add consumers to or remove consumers from a datashare. To do so, they use `GRANT USAGE` or `REVOKE USAGE`.

To find the namespace of the cluster that you currently see, you can use the `SELECT CURRENT_NAMESPACE` command. To find the namespace of a different cluster within the same AWS

account, go to the Amazon Redshift console cluster details page. On that page, find the newly added namespace field.

You can also use the Amazon Redshift console to add or remove data consumers for datashares. For more information, see [Adding data consumers to datashares \(p. 358\)](#) and [Removing data consumers from datashares \(p. 362\)](#).

5. (Optional) Add security restrictions to the datashare. The following example shows that the consumer cluster with a public IP access is allowed to read the datashare. For more information, see [ALTER DATASHARE \(p. 629\)](#).

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE = TRUE;
```

You can modify properties about the type of consumers after datashare creation. For example, you can define that clusters that want to consume data from a given datashare can't be publicly accessible. Queries from consumer clusters that don't meet security restrictions specified in datashare are rejected at query runtime.

You can also use the Amazon Redshift console to edit datashares. For more information, see [Editing datashares created in your account \(p. 362\)](#).

6. List datashares created in the cluster and look into the contents of the datashare.

The following example displays the information of a datashare named `salesshare`. For more information, see [DESC DATASHARE \(p. 860\)](#) and [SHOW DATASHARES \(p. 986\)](#).

```
DESC DATASHARE salesshare;

producer_account | producer_namespace | share_type | share_name |
object_type | object_name | include_new |
-----+-----+-----+-----+
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salesshare |
table | public.ticket_users_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salesshare |
table | public.ticket_venue_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salesshare |
table | public.ticket_category_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salesshare |
table | public.ticket_date_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salesshare |
table | public.ticket_event_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salesshare |
table | public.ticket_listing_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salesshare |
table | public.ticket_sales_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salesshare |
schema | public | t |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salesshare |
view | public.sales_data_summary_view |
```

The following example displays the outbound datashares in a producer cluster.

```
SHOW DATASHARES LIKE 'sales%';
```

The output looks similar to the following.

| share_name | share_owner | source_database | consumer_database | share_type | createdate          | is_publicaccessible | share_acl | producer_account | producer_namespace                   |
|------------|-------------|-----------------|-------------------|------------|---------------------|---------------------|-----------|------------------|--------------------------------------|
| salesshare | 100         | dev             |                   | OUTBOUND   | 2020-12-09 02:27:08 | True                |           | 123456789012     | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |

For more information, see [DESC DATASHARE \(p. 860\)](#) and [SHOW DATASHARES \(p. 986\)](#).

You can also use [SVV\\_DATASHARES \(p. 1422\)](#), [SVV\\_DATASHARE\\_CONSUMERS \(p. 1423\)](#), and [SVV\\_DATASHARE\\_OBJECTS \(p. 1424\)](#) to view the datashares, the objects within the datashare, and the datashare consumers.

7. Drop datashares. For more information, see [DROP DATASHARE \(p. 864\)](#).

You can delete the datashare objects at any point using [DROP DATASHARE \(p. 864\)](#). Cluster superusers and owners of datashare can drop datashares.

The following example drops a datashare named salesshare.

```
DROP DATASHARE salesshare;
```

You can also use the Amazon Redshift console to delete datashares. For more information, see [Deleting datashares created in your account \(p. 363\)](#).

8. Use ALTER DATASHARE to remove objects from datashares at any point from the datashare. Use REVOKE USAGE ON to revoke permissions on the datashare to certain consumers. It revokes USAGE permissions on objects within a datashare and instantly stops access to all consumer clusters. Listing datashares and the metadata queries, such as listing databases and tables, doesn't return the shared objects after access is revoked.

```
ALTER DATASHARE salesshare REMOVE TABLE public.ticket_sales_redshift;
```

You can also use the Amazon Redshift console to edit datashares. For more information, see [Editing datashares created in your account \(p. 362\)](#).

9. Revoke access to the datashare from namespaces if you don't want to share the data with the consumers anymore.

```
REVOKE USAGE ON DATASHARE salesshare FROM NAMESPACE
'13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

You can also use the Amazon Redshift console to edit datashares. For more information, see [Editing datashares created in your account \(p. 362\)](#).

### To share data for read purposes as a consumer cluster administrator

1. List the datashares that are made available to you and view the content of datashares. For more information, see [DESC DATASHARE \(p. 860\)](#) and [SHOW DATASHARES \(p. 986\)](#).

The following example displays the information of inbound datashares of a specified producer namespace. When you run DESC DATASHARE as a consumer cluster administrator, you must specify the NAMESPACE option to view inbound datashares.

```
DESC DATASHARE salesshare OF NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';

producer_account | producer_namespace | share_type | share_name |
object_type | object_name | include_new
-----+-----+-----+-----+
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_users_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_venue_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_category_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_date_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_event_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_listing_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_sales_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
schema | public |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
view | public.sales_data_summary_view |
```

Only cluster superusers can do this. You can also use SVV\_DATASHARES to view the datashares and SVV\_DATASHARE\_OBJECTS to view the objects within the datashare.

The following example displays the inbound datashares in a consumer cluster.

```
SHOW DATASHARES LIKE 'sales%';

share_name | share_owner | source_database | consumer_database | share_type |
| createdate | is_publicaccessible | share_acl | producer_account |
producer_namespace
-----+-----+-----+-----+
+-----+-----+-----+
+-----+
salesshare | | | | | INBOUND
| | | | | 123456789012 |
13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

- As a database superuser, you can create local databases that reference to the datashares. For more information, see [CREATE DATABASE \(p. 745\)](#).

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF NAMESPACE
'13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

You can see databases that you created from the datashare by querying the [SVV\\_REDSHIFT\\_DATABASES \(p. 1445\)](#) view. You can't connect to these databases created from datashares, and they are read-only. However, you can connect to a local database on your consumer cluster and perform a cross-database query to query the data from the databases created from datashares. You can't create a datashare on top of database objects created from an existing datashare. However, you can copy the data into a separate table on the consumer cluster, perform any processing needed, and then share the new objects that were created.

You can also use the Amazon Redshift console to create databases from datashares. For more information, see [Creating databases from datashares \(p. 362\)](#).

3. (Optional) Create external schemas to refer to and assign granular permissions to specific schemas in the consumer database imported on the consumer cluster. For more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#).

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA 'public';
```

4. Grant permissions on databases and schema references created from the datashares to users or groups in the consumer cluster as needed. For more information, see [GRANT \(p. 887\)](#) or [REVOKE \(p. 916\)](#).

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO GROUP Analyst_group;
```

As a consumer cluster administrator, you can only assign permissions on the entire database created from the datashare to your users and groups. In some cases, you need fine-grained controls on a subset of database objects created from the datashare. If so, you can create an external schema reference that points to specific schemas in the datashare (as described in the previous step) and provide granular permissions at schema level.

You can also create late-binding views on top of shared objects and use these to assign granular permissions. You can also consider having producer clusters create additional datashares for you with the granularity required.

5. Query data in the shared objects in the datashares.

Users and groups with permissions on consumer databases and schemas on consumer clusters can explore and navigate the metadata of any shared objects. They can also explore and navigate local objects in a consumer cluster. To do this, they use JDBC or ODBC drivers or SVV\_ALL and SVV\_REDSHIFT views.

Producer clusters might have many schemas in the database, tables, and views within each schema. The users on the consumer side can see only the subset of objects that are made available through the datashare. These users can't see the entire metadata from the producer cluster. This approach helps provide granular metadata security control with data sharing.

You continue to connect to local cluster databases. But now, you can also read from the databases and schemas that are created from the datashare using the three-part database.schema.table notation. You can perform queries that span across any and all databases that are visible to you. These can be local databases on the cluster or databases created from the datashares. Consumer clusters can't connect to the databases created from the datashares.

You can access the data using full qualification. For more information, see [Examples of using a cross-database query \(p. 312\)](#).

```
SELECT * FROM sales_db.public.ticket_sales_redshift ORDER BY 1,2 LIMIT 5;

salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
commission | saletime
-----+-----+-----+-----+-----+-----+-----+
+-----+
1 | 1 | 36861 | 21191 | 7872 | 1875 | 4 | 728.00 |
109.20 | 2008-02-18 02:36:48
2 | 4 | 8117 | 11498 | 4337 | 1983 | 2 | 76.00 |
11.40 | 2008-06-06 05:00:16
3 | 5 | 1616 | 17433 | 8647 | 1983 | 2 | 350.00 |
52.50 | 2008-06-06 08:26:17
4 | 5 | 1616 | 19715 | 8647 | 1986 | 1 | 175.00 |
26.25 | 2008-06-09 08:38:52
```

|       |                     |       |       |      |      |   |        |
|-------|---------------------|-------|-------|------|------|---|--------|
| 5     | 6                   | 47402 | 14115 | 8240 | 2069 | 2 | 154.00 |
| 23.10 | 2008-08-31 09:17:02 |       |       |      |      |   |        |

You can only use SELECT statements on shared objects. However, you can create tables in the consumer cluster by querying the data from the shared objects in a different local database.

In addition to queries, consumers can create views on shared objects. Only late-binding views or materialized views are supported. Amazon Redshift doesn't support regular views on shared data. Views that consumers create can span across multiple local databases or databases created from datashares. For more information, see [CREATE VIEW \(p. 851\)](#).

```
// Connect to a local cluster database
// Create a view on shared objects and access it.
CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.ticket_sales_redshift
WITH NO SCHEMA BINDING;
SELECT * FROM sales_data;
```

## Sharing data across AWS accounts

You can share data for read purposes across AWS accounts. Sharing data across AWS accounts works similarly to sharing data within an account. The difference is that there is a two-way handshake required in sharing data across AWS accounts. A producer account administrators can either authorize consumer accounts to access datashares or choose not to authorize any access. To use an authorized datashare, a consumer account administrator can associate the datashare. The administrator can associate the datashare with an entire AWS account or with specific clusters in the consumer account, or decline the datashare. For more information about sharing data within an account, see [Sharing data within an AWS account \(p. 333\)](#).

A datashare can have data consumers that are either cluster namespaces in the same account or different AWS accounts. You don't need to create separate datashares for sharing within an account and cross-account sharing.

For cross-account data sharing, both the producer and consumer cluster must be encrypted.

When sharing data with AWS accounts, producer cluster administrators share with the AWS account as an entity. A consumer cluster administrator can decide which cluster namespaces in the consumer account get access to a datashare.

### Topics

- [Producer cluster administrator actions \(p. 339\)](#)
- [Consumer account administrator actions \(p. 341\)](#)
- [Consumer cluster administrator actions \(p. 341\)](#)

## Producer cluster administrator actions

If you are a producer cluster administrator or database owner – follow these steps:

1. Create datashares in your cluster and add datashare objects to the datashares. For more detailed steps on how to create datashares and add datashare objects to datashares, see [Sharing data within an AWS account \(p. 333\)](#). For information about the CREATE DATASHARE and ALTER DATASHARE, see [CREATE DATASHARE \(p. 754\)](#) and [ALTER DATASHARE \(p. 629\)](#).

The following example adds different datashare objects to the datashare `salesshare`.

```
-- Add schema to datashare
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;

-- Add table under schema to datashare
ALTER DATASHARE salesshare ADD TABLE public.ticket_sales_redshift;

-- Add view to datashare
ALTER DATASHARE salesshare ADD TABLE public.sales_data_summary_view;

-- Add all existing tables and views under schema to datashare (does not include future
-- table)
ALTER DATASHARE salesshare ADD ALL TABLES in schema public;
```

You can also use the Amazon Redshift console to create or edit datashares. For more information, see [Creating datashares \(p. 357\)](#) and [Editing datashares created in your account \(p. 362\)](#).

2. Delegate permissions to operate on the datashare. For more information, see [GRANT \(p. 887\)](#) or [REVOKE \(p. 916\)](#).

The following example grants permissions to `dbuser` on `salesshare`.

```
GRANT ALTER, SHARE ON DATASHARE salesshare TO dbuser;
```

Cluster superusers and the owners of the datashare can grant or revoke modification permissions on the datashare to additional users.

3. Add consumers to or remove consumers from datashares. The following example adds the AWS account ID to `salesshare`. For more information, see [GRANT \(p. 887\)](#) or [REVOKE \(p. 916\)](#).

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '123456789012';
```

You can only grant permissions to one data consumer in a GRANT statement.

Cluster superusers and the owners of datashare objects, or users that have SHARE permissions on the datashare, can add consumers to or remove consumers from a datashare. To do so, they use GRANT USAGE or REVOKE USAGE.

You can also use the Amazon Redshift console to add or remove data consumers for datashares. For more information, see [Adding data consumers to datashares \(p. 358\)](#) and [Removing data consumers from datashares \(p. 362\)](#).

4. (Optional) Revoke access to the datashare from AWS accounts if you don't want to share the data with the consumers anymore.

```
REVOKE USAGE ON DATASHARE salesshare FROM ACCOUNT '123456789012';
```

**If you are a producer account administrator** – follow these steps:

After granting usage to the AWS account, the datashare status is pending\_authorization. The producer account administrator should authorize datashares using the Amazon Redshift console and choose the data consumers.

Sign in to the <https://console.aws.amazon.com/redshift/>. Then choose which data consumers to authorize to access datashares or to remove authorization from. Authorized data consumers receive notifications to take actions on datashares. If you are adding a cluster namespace as a data consumer, you don't have to perform authorization. After data consumers are authorized, they can access datashare

objects and create a consumer database to query the data. For more information, see [Authorizing or removing authorization from datashares \(p. 359\)](#).

## Consumer account administrator actions

**If you are a consumer account administrator** – follow these steps:

To associate one or more datashares that are shared from other accounts with your entire AWS account or specific cluster namespaces in your account, use the Amazon Redshift console.

Sign in to the <https://console.aws.amazon.com/redshift/>. Then associate one or more datashares that are shared from other accounts with your entire AWS account or specific cluster namespaces in your account. For more information, see [Associating datashares \(p. 359\)](#).

After the AWS account or specific cluster namespaces are associated, the datashares become available for consumption. You can also change datashare association at any time. When changing association from individual cluster namespaces to an AWS account, Amazon Redshift overwrites the cluster namespaces with the AWS account information. When changing association from an AWS account to specific cluster namespaces, Amazon Redshift overwrites the AWS account information with the cluster namespace information. All cluster namespaces in the account get access to the data.

## Consumer cluster administrator actions

**If you are a consumer cluster administrator** – follow these steps:

1. List the datashares made available to you and view the content of datashares. The content of datashares is available only when the producer cluster administrator has authorized the datashares and the consumer cluster administrator has accepted and associated the datashares. For more information, see [DESC DATASHARE \(p. 860\)](#) and [SHOW DATASHARES \(p. 986\)](#).

The following example displays the information of inbound datashares of a specified producer namespace. When you run the DESC DATAHSARE as a consumer cluster administrator, you must specify the NAMESPACE and account ID to view inbound datashares. For outbound datashares, specify the datashare name.

```
SHOW DATASHARES LIKE 'sales%';

share_name | share_owner | source_database | consumer_database | share_type | createdate
| is_publicaccessible | share_acl | producer_account | producer_namespace
-----+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+
salesshare | t | | | 123456789012 | INBOUND | 'dd8772e1-
d792-4fa4-996b-1870577efc0d'
```

```
DESC DATASHARE salesshare OF ACCOUNT '123456789012' NAMESPACE 'dd8772e1-
d792-4fa4-996b-1870577efc0d';
```

```
producer_account | producer_namespace | share_type | share_name |
object_type | object_name
-----+-----+-----+-----+
+-----+-----+-----+-----+
123456789012 | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND | salesshare |
table | public.ticket_users_redshift
123456789012 | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND | salesshare |
table | public.ticket_venue_redshift
123456789012 | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND | salesshare |
table | public.ticket_category_redshift
```

```

123456789012 | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND | salesshare |
table | public.ticket_date_redshift
123456789012 | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND | salesshare |
table | public.ticket_event_redshift
123456789012 | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND | salesshare |
table | public.ticket_listing_redshift
123456789012 | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND | salesshare |
table | public.ticket_sales_redshift
123456789012 | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND | salesshare |
schema | public
(8 rows)

```

Only cluster superusers can do this. You can also use `SVV_DATASHARES` to view the datashares and `SVV_DATASHARE_OBJECTS` to view the objects within the datashare.

The following example displays the inbound datashares in a consumer cluster.

```

SELECT * FROM SVV_DATASHARES WHERE share_name LIKE 'sales%';

share_name | share_owner | source_database | consumer_database | share_type | createdate
| is_publicaccessible | share_acl | producer_account | producer_namespace
-----+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+
salesshare | | | | 123456789012 | INBOUND |
| t | | | 'dd8772e1-
d792-4fa4-996b-1870577efc0d'

```

```

SELECT * FROM SVV_DATASHARE_OBJECTS WHERE share_name LIKE 'sales%';

share_type | share_name | object_type | object_name |
producer_account | producer_namespace
-----+-----+-----+-----+
+-----+
INBOUND | salesshare | table | public.ticket_users_redshift | 123456789012
| dd8772e1-d792-4fa4-996b-1870577efc0d
INBOUND | salesshare | table | public.ticket_venue_redshift | 123456789012
| dd8772e1-d792-4fa4-996b-1870577efc0d
INBOUND | salesshare | table | public.ticket_category_redshift | 123456789012
| dd8772e1-d792-4fa4-996b-1870577efc0d
INBOUND | salesshare | table | public.ticket_date_redshift | 123456789012
| dd8772e1-d792-4fa4-996b-1870577efc0d
INBOUND | salesshare | table | public.ticket_event_redshift | 123456789012
| dd8772e1-d792-4fa4-996b-1870577efc0d
INBOUND | salesshare | table | public.ticket_listing_redshift | 123456789012
| dd8772e1-d792-4fa4-996b-1870577efc0d
INBOUND | salesshare | table | public.ticket_sales_redshift | 123456789012
| dd8772e1-d792-4fa4-996b-1870577efc0d
INBOUND | salesshare | schema | public
| dd8772e1-d792-4fa4-996b-1870577efc0d
(8 rows)

```

2. Create local databases that reference to the datashares. Specify the `NAMESPACE` and account ID when creating the database from the datashare. For more information, see [CREATE DATABASE \(p. 745\)](#).

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF ACCOUNT '123456789012' NAMESPACE
'dd8772e1-d792-4fa4-996b-1870577efc0d';
```

You can see databases that you created from the datashare by querying [SVV\\_REDSHIFT\\_DATABASES \(p. 1445\)](#) view. You can't connect to these databases created from datashares, and they are read-only. However, you can connect to a local database on your consumer cluster and perform a cross-database query on the data from the databases created from datashares.

You can't create a datashare on top of database objects created from an existing datashare. However, you can copy the data into a separate table on the consumer cluster, perform any processing needed, and then share the new objects created.

3. (Optional) Create external schemas to refer and assign granular permissions to specific schemas in the consumer database imported on the consumer cluster. For more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#).

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA 'public';
```

4. Grant permissions on databases and schema references created from the datashares to users groups in the consumer cluster as needed. For more information, see [GRANT \(p. 887\)](#) or [REVOKE \(p. 916\)](#).

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO GROUP Analyst_group;
```

As a consumer cluster administrator, you can only assign permissions on the entire database created from the datashare to your users and groups. In some cases, you need fine-grained controls on a subset of database objects created from the datashare. If so, you can create an external schema reference pointing to specific schemas in the datashare, as described in the previous step. You can then provide granular permissions at the schema level. You can also create late-binding views on top of shared objects and use these to assign granular permissions. You can also consider having producer clusters create additional datashares for you with the granularity required. You can create as many schema references to the database created from the datashare.

5. Query data in the shared objects in the datashares.

Users and groups with permissions on consumer databases and schemas on consumer clusters can explore and navigate the metadata of any shared objects. They can also explore and navigate local objects in a consumer cluster. To do this, use JDBC or ODBC drivers or SVV\_ALL and SVV\_REDSHIFT views.

Producer clusters might have many schemas in the database, tables, and views within each schema. The users on the consumer side can see only the subset of objects that are made available through the datashare. These users can't see all the metadata from the producer cluster. This approach helps provide granular metadata security control with data sharing.

You continue to connect to local cluster databases. But now, you can also read from the databases and schemas that are created from the datashare using the three-part database.schema.table notation. You can perform queries that span across any and all databases that are visible to you. These can be local databases on the cluster or databases created from the datashares. Consumer clusters can't connect to the databases created from the datashares.

You can access the data using full qualification. For more information, see [Examples of using a cross-database query \(p. 312\)](#).

```
SELECT * FROM sales_db.public.ticket_sales_redshift;
```

You can only use SELECT statements on shared objects. However, you can create tables in the consumer cluster by querying the data from the shared objects in a different local database.

In addition to performing queries, consumers can create views on shared objects. Only late-binding views and materialized views are supported. Amazon Redshift doesn't support regular views on shared data. Views that consumers create can span across multiple local databases or databases created from datashares. For more information, see [CREATE VIEW \(p. 851\)](#).

```
// Connect to a local cluster database

// Create a view on shared objects and access it.
CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.ticket_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

## Sharing data across AWS Regions

You can share data for read purposes across Amazon Redshift clusters in AWS Regions. With cross-Region data sharing, you can share data across AWS Regions without the need to copy data manually. You don't have to unload your data into Amazon S3 and copy the data into a new Amazon Redshift cluster or perform cross-Region snapshot copy.

With cross-Region data sharing, you can share data across clusters in the same AWS account, or in different AWS accounts even when the clusters are in different Regions. When sharing data with Amazon Redshift clusters that are in the same AWS account but different AWS Regions, follow the same workflow as sharing data within an AWS account. For more information, see [Sharing data within an AWS account \(p. 333\)](#).

If clusters sharing data are in different AWS accounts and AWS Regions, you can follow the same workflow as sharing data across AWS accounts and include Region-level associations on the consumer cluster. Cross-Region data sharing supports datashare association with the entire AWS account, the entire AWS Region, or specific cluster namespaces within an AWS Region. For more information about sharing data across AWS accounts, see [Sharing data across AWS accounts \(p. 339\)](#).

To share data between different AWS Regions, you pay the cross-Region data transfer from the producer Region to the consumer Region based on the price of Amazon S3 in addition to the cost of using your Amazon Redshift cluster.

To use the datashare, a consumer account administrator can associate the datashare in one of the following three ways.

- Association with an entire AWS account spanning all its AWS Regions
- Association with a specific AWS Region in an AWS account
- Association with specific cluster namespaces within an AWS Region

When the administrator chooses the entire AWS account, all existing and future cluster namespaces across different AWS Regions in the account have access to the datashares. A consumer account administrator can also choose specific AWS Regions or cluster namespaces within a Region to grant them access to the datashares.

**If you are a producer cluster administrator or database owner**, create a datashare, add database objects and data consumers to the datashare, and grant permissions to data consumers. For more information, see [Producer cluster administrator actions \(p. 339\)](#).

**If you are a producer account administrator**, authorize datashares using the AWS Command Line Interface (AWS CLI) or the Amazon Redshift console and choose the data consumers.

**If you are a consumer account administrator** – follow these steps:

To associate one or more datashares that are shared from other accounts to your entire AWS account or specific AWS Regions or cluster namespaces within an AWS Region, use the Amazon Redshift console.

With cross-Region datasharing, you can add clusters in a specific AWS Region using the AWS Command Line Interface (AWS CLI) or Amazon Redshift console.

To specify one or more AWS Regions, you can use the `associate-data-share-consumer` CLI command with the optional `consumer-region` option.

With the CLI, the following example associates the `Salesshare` with the entire AWS account with the `associate-whole-account` option. You can only associate one Region at a time.

```
aws redshift associate-data-share-consumer
--region {PRODUCER_REGION}
--data-share-arn arn:aws:redshift:{PRODUCER_REGION}:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/Salesshare
--associate-whole-account
```

The following example associates the `Salesshare` with the US East (Ohio) Region (`us-east-2`).

```
aws redshift associate-data-share-consumer
--region {PRODUCER_REGION}
--data-share-arn arn:aws:redshift:{PRODUCER_REGION}:0123456789012:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/Salesshare
--consumer-region 'us-east-2'
```

The following example associates the `Salesshare` with a specific consumer cluster namespace in another AWS account in the Asia Pacific (Sydney) Region (`ap-southeast-2`).

```
aws redshift associate-data-share-consumer
--data-share-arn arn:aws:redshift:{PRODUCER_REGION}:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/Salesshare
--consumer-arn 'arn:aws:redshift:ap-southeast-2:{CONSUMER_ACCOUNT}:namespace:
{ConsumerImmutableClusterId}'
```

You can use the Amazon Redshift console to associate datashares with your entire AWS account or specific AWS Regions or cluster namespaces within an AWS Region. To do this, sign in to the <https://console.aws.amazon.com/redshift/>. Then associate one or more datashares that are shared from other accounts with your entire AWS account, the entire AWS Region, or a specific cluster namespace within an AWS Region. For more information, see [Associating datashares \(p. 359\)](#).

After the AWS account or specific cluster namespaces are associated, the datashares become available for consumption. You can also change datashare association at any time. When changing association from individual cluster namespaces to an AWS account, Amazon Redshift overwrites the cluster namespaces with the AWS account information. When changing association from an AWS account to specific cluster namespaces, Amazon Redshift overwrites the AWS account information with the cluster namespace information. When changing association from an entire AWS account to specific AWS Regions and cluster namespaces, Amazon Redshift overwrites the AWS account information with the specific Region and cluster namespace information.

If you are a **consumer cluster administrator**, you can create local databases that reference to the datashares and grant permissions on databases created from the datashares to users groups in the consumer cluster as needed. You can also create views on shared objects and create external schemas to refer and assign granular permissions to specific schemas in the consumer database imported on the consumer cluster. For more information, see [Consumer cluster administrator actions \(p. 341\)](#).

## Managing cost control for cross-Region data sharing

To share data between different AWS Regions, you pay the cross-Region data transfer from the producer Region to the consumer Region in addition to the cost of using your Amazon Redshift cluster. The price of data transfer is different for different Regions. The charge is based on the bytes of data scanned for

every successful query run. For more information about Amazon Redshift pricing, see [Amazon Redshift pricing](#).

You are charged for the number of bytes, rounded up to the next megabyte, with a 10MB minimum per query. You can set cost controls on your query usage and view the amount of data being transferred per query on your cluster.

To monitor and control your usage and associated cost of using cross-Region data sharing, you can create daily, weekly, monthly usage limits, and define actions that Amazon Redshift automatically takes if those limits are reached to help maintain your budget with predictability. For more information about usage limits in Amazon Redshift, see [Managing usage limits in Amazon Redshift](#).

Depending on the usage limits you set, actions that Amazon Redshift takes can be to log an event to a system table, send a CloudWatch alarm and notify an administrator with an Amazon SNS, or to turn off cross-Region data sharing for further usage. For more information about the actions, see [Managing usage limits in Amazon Redshift](#).

To create usage limits in the Amazon Redshift console, choose **Configure usage limit** under **Actions** for your cluster. You can monitor your usage trends and get alerts on usage exceeding your defined limits with automatically generated CloudWatch metrics from the **Cluster performance** or **Monitoring** tabs. You can also create, modify, and delete usage limits programmatically by using the AWS CLI or Amazon Redshift API operations. For more information, see [Managing usage limits in Amazon Redshift](#).

## Sharing licensed Amazon Redshift data on AWS Data Exchange

When creating AWS Data Exchange datashares and adding them to an AWS Data Exchange product, providers can license data in Amazon Redshift that consumers can discover, subscribe to, and query up-to-date data in Amazon Redshift when they have active AWS Data Exchange subscriptions.

With AWS Data Exchange datashares added to an AWS Data Exchange product, consumers automatically have access to a product's datashares when their subscription starts and retain their access as long as their subscription is active.

### Working with AWS Data Exchange datashares as a producer

**If you are a producer cluster administrator, follow these steps to manage AWS Data Exchange datashares on the Amazon Redshift console:**

1. Create datashares in your cluster to share data on AWS Data Exchange and grant access to AWS Data Exchange to the datashares.

Cluster superuser and database owners can create datashares. Each datashare is associated with a database during creation. Only objects from that database can be shared in that datashare. Multiple datashares can be created on the same database with the same or different granularity of objects. There is no limit on the number of datashares you can create on a cluster.

You can also use the Amazon Redshift console to create datashares. For more information, see [Creating datashares \(p. 357\)](#).

Use the MANAGEDBY ADX option to implicitly grant access of the datashare to AWS Data Exchange when running the CREATE DATASHARE statement. This indicates that AWS Data Exchange manages this datashare. You can only use the MANAGEDBY ADX option when you create a new datashare. You can't use the ALTER DATASHARE statement to modify an existing datashare to add the MANAGEDBY ADX option. Once a datashare is created with the MANAGEDBY ADX option, only AWS Data Exchange can access and manage the datashare.

```
CREATE DATASHARE salesshare
[[SET] MANAGEDBY [=] {ADX}];
```

2. Add objects to the datashares. Producer administrator continues to manage datashare objects that are available in an AWS Data Exchange datashare.

To add objects to a datashare, add the schema before adding objects. When you add a schema, Amazon Redshift doesn't add all the objects under it. You must add them explicitly. For more information, see [ALTER DATASHARE \(p. 629\)](#).

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare ADD TABLE public.ticket_sales_redshift;
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

You can also add views to a datashare.

```
CREATE VIEW public.sales_data_summary_view AS SELECT * FROM
public.ticket_sales_redshift;
ALTER DATASHARE salesshare ADD TABLE public.sales_data_summary_view;
```

Use ALTER DATASHARE to share schemas, and tables, views, and functions in a given schema. Superusers, datashare owners, or users who have ALTER or ALL permissions on the datashare can alter the datashare to add objects to or remove objects from it. Users should have the permissions to add or remove objects from the datashare. Users should also be the owners of the objects or have SELECT, USAGE, or ALL permissions on the objects.

Use the INCLUDENEW clause to add any new tables, views, or SQL user-defined functions (UDFs) created in a specified schema to the datashare. Only superusers can change this property for each datashare-schema pair.

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare SET INCLUDENEW = TRUE FOR SCHEMA PUBLIC;
```

You can also use the Amazon Redshift console to add or remove objects from datashares. For more information, see [Adding datashare objects to datashares \(p. 358\)](#), [Removing datashare objects from datashares \(p. 361\)](#), and [Editing AWS Data Exchange datashares \(p. 363\)](#).

3. To authorize access to the datashares for AWS Data Exchange, do one of the following:

- Explicitly authorize access to the datashare for AWS Data Exchange by using the ADX keyword in the aws redshift authorize-data-share API. This allows AWS Data Exchange to recognize the datashare in the service account and manage associating consumers to the datashare.

```
aws redshift authorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier ADX
```

You can use a conditional key ConsumerIdentifier for the AuthorizeDataShare and DeauthorizeDataShare APIs to explicitly allow or deny AWS Data Exchange to make calls to the two APIs in the IAM policy.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "VisualEditor0",
 "Effect": "Deny",
 "Action": [
 "redshift:AuthorizeDataShare",
 "redshift:DeauthorizeDataShare"
]
 }
]
}
```

```

],
 "Resource": "*",
 "Condition": {
 "StringEqualsIgnoreCase": {
 "redshift:ConsumerIdentifier": "ADX"
 }
 }
]
}

```

- Use the Amazon Redshift console to authorize or remove authorization of AWS Data Exchange datashares. For more information, see [Authorizing or removing authorization from datashares \(p. 359\)](#).
- Optionally, you can implicitly authorize access to the AWS Data Exchange datashare when importing the datashare into an AWS Data Exchange dataset.

To remove authorization for access to the AWS Data Exchange datashares, use the ADX keyword in the `aws redshift deauthorize-data-share` API operation. By doing this, you allow AWS Data Exchange to recognize the datashare in the service account and manage removing association from the datashare.

```
aws redshift deauthorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier ADX
```

4. List datashares created in the cluster and look into the contents of the datashare.

The following example displays the information of a datashare named salessshare. For more information, see [DESC DATASHARE \(p. 860\)](#) and [SHOW DATASHARES \(p. 986\)](#).

```
DESC DATASHARE salessshare;

producer_account | producer_namespace | share_type | share_name |
object_type | object_name | include_new
-----+-----+-----+-----+
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salessshare |
table | public.ticket_users_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salessshare |
table | public.ticket_venue_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salessshare |
table | public.ticket_category_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salessshare |
table | public.ticket_date_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salessshare |
table | public.ticket_event_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salessshare |
table | public.ticket_listing_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salessshare |
table | public.ticket_sales_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salessshare |
schema | public | t |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salessshare |
view | public.sales_data_summary_view |
```

The following example displays the outbound datashares in a producer cluster.

```
SHOW DATASHARES LIKE 'sales%';
```

The output looks similar to the following.

| share_name | share_owner | source_database | consumer_database | share_type | createdate                           | is_publicaccessible | share_acl | producer_account | producer_namespace |
|------------|-------------|-----------------|-------------------|------------|--------------------------------------|---------------------|-----------|------------------|--------------------|
| salesshare | 100         | dev             |                   |            | 2020-12-09 02:27:08                  | True                |           | 123456789012     |                    |
|            |             |                 |                   |            | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |                     |           |                  |                    |

For more information, see [DESC DATASHARE \(p. 860\)](#) and [SHOW DATASHARES \(p. 986\)](#).

You can also use [SVV\\_DATASHARES \(p. 1422\)](#), [SVV\\_DATASHARE\\_CONSUMERS \(p. 1423\)](#), and [SVV\\_DATASHARE\\_OBJECTS \(p. 1424\)](#) to view the datashares, the objects within the datashare, and the datashare consumers.

- Drop datashares. We recommend that you don't delete an AWS Data Exchange datashare shared to other AWS accounts using the `DROP DATASHARE` statement. Those accounts will lose access to the datashare. This action is irreversible. This might breach data product offer terms in AWS Data Exchange. If you want to delete an AWS Data Exchange datashare, see [DROP DATASHARE usage notes \(p. 864\)](#).

The following example drops a datashare named `salesshare`.

```
DROP DATASHARE salesshare;
ERROR: Drop of ADX-managed datashare salesshare requires session variable
datashare_break_glass_session_var to be set to value '620c871f890c49'
```

To allow dropping an AWS Data Exchange datashare, set the `datashare_break_glass_session_var` variable and run the `DROP DATASHARE` statement again. If you want to delete an AWS Data Exchange datashare, see [DROP DATASHARE usage notes \(p. 864\)](#).

You can also use the Amazon Redshift console to delete datashares. For more information, see [Deleting AWS Data Exchange datashares created in your account \(p. 364\)](#).

- Use `ALTER DATASHARE` to remove objects from datashares at any point from the datashare. Use `REVOKE USAGE ON` to revoke permissions on the datashare to certain consumers. It revokes `USAGE` privileges on objects within a datashare and instantly stops access to all consumer clusters. Listing datashares and the metadata queries, such as listing databases and tables, doesn't return the shared objects after access is revoked.

```
ALTER DATASHARE salesshare REMOVE TABLE public.ticket_sales_redshift;
```

You can also use the Amazon Redshift console to edit datashares. For more information, see [Editing AWS Data Exchange datashares \(p. 363\)](#).

- Grant or revoke `GRANT USAGE` from AWS Data Exchange datashares. You can't grant or revoke `GRANT USAGE` for AWS Data Exchange datashare. The following example shows an error when the `GRANT USAGE` permission is granted to an AWS account for a datashare that AWS Data Exchange manages.

```
CREATE DATASHARE salesshare MANAGEDBY ADX;
```

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '012345678910';
```

ERROR: Permission denied to add/remove consumer to/from datashare salessshare.  
Datashare consumers are managed by ADX.

For more information, see [GRANT \(p. 887\)](#) or [REVOKE \(p. 916\)](#).

**If you are a producer cluster administrator, follow these steps to create and publish a datashare product on the AWS Data Exchange console:**

- When the AWS Data Exchange datashare has been created, the producer creates a new dataset, imports assets, creates a revision, and creates and publishes a new product.

Use the Amazon Redshift console to create datasets. For more information, see [Creating data sets on AWS Data Exchange \(p. 363\)](#).

For more information, see [Providing data products on AWS Data Exchange](#).

## Working with AWS Data Exchange datashares as a consumer

**If you are a consumer, follow these steps to discover data products that contain AWS Data Exchange datashares and query Amazon Redshift data:**

- On the AWS Data Exchange console, discover and subscribe to data products that contains AWS Data Exchange datashares.

Once your subscription starts, you can access licensed Amazon Redshift data that is imported as assets to datasets that contain AWS Data Exchange datashares.

For more information on how to get started with using data products that contain AWS Data Exchange datashares, see [Subscribing to data products on AWS Data Exchange](#).

- On the Amazon Redshift console, create an Amazon Redshift cluster, if needed.

For information on how to create a cluster, see [Creating a cluster](#).

- List the datashares that are made available to you and view the content of datashares. For more information, see [DESC DATASHARE \(p. 860\)](#) and [SHOW DATASHARES \(p. 986\)](#).

The following example displays the information of inbound datashares of a specified producer namespace. When you run DESC DATASHARE as a consumer cluster administrator, you must specify the ACCOUNT and NAMESPACE option to view inbound datashares.

```
DESC DATASHARE salesshare of ACCOUNT '123456789012' NAMESPACE
'13b8833d-17c6-4f16-8fe4-1a018f5ed00d';

producer_account | producer_namespace | share_type | share_name |
object_type | object_name | include_new
-----+-----+-----+-----+
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_users_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_venue_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_category_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_date_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_event_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_listing_redshift |
```

```

123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
table | public.ticket_sales_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
schema | public |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare |
view | public.sales_data_summary_view |

```

Only cluster superusers can do this. You can also use SVV\_DATASHARES to view the datashares and SVV\_DATASHARE\_OBJECTS to view the objects within the datashare.

The following example displays the inbound datashares in a consumer cluster.

```

SHOW DATASHARES LIKE 'sales%';

share_name | share_owner | source_database | consumer_database | share_type
| createdate | is_publicaccessible | share_acl | producer_account |
producer_namespace
-----+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+
salesshare | | t | | INBOUND
| | | | | 123456789012 |
13b8833d-17c6-4f16-8fe4-1a018f5ed00d

```

4. Create local databases that reference to the datashares. You must specify the ACCOUNT and NAMESPACE option to create local databases for AWS Data Exchange datashares. For more information, see [CREATE DATABASE \(p. 745\)](#).

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF ACCOUNT '123456789012' NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

You can see databases that you created from the datashare by querying the [SVV\\_REDSHIFT\\_DATABASES \(p. 1445\)](#) view. You can't connect to these databases created from datashares, and they are read-only. However, you can connect to a local database on your consumer cluster and perform a cross-database query on the data from the databases created from datashares. You can't create a datashare on top of database objects created from an existing datashare. However, you can copy the data into a separate table on the consumer cluster, perform any processing needed, and then share the new objects that were created.

You can also use the Amazon Redshift console to create databases from datashares. For more information, see [Creating databases from datashares \(p. 362\)](#).

5. (Optional) Create external schemas to refer to and assign granular permissions to specific schemas in the consumer database imported on the consumer cluster. For more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#).

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA 'public';
```

6. Grant permissions on databases and schema references created from the datashares to users or groups in the consumer cluster as needed. For more information, see [GRANT \(p. 887\)](#) or [REVOKE \(p. 916\)](#).

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO GROUP Analyst_group;
```

As a consumer cluster administrator, you can only assign permissions on the entire database created from the datashare to your users and groups. In some cases, you need fine-grained controls on a subset of database objects created from the datashare. If so, you can create an external schema reference that points to specific schemas in the datashare (as described in the previous step) and provide granular permissions at schema level.

You can also create late-binding views on top of shared objects and use these to assign granular permissions. You can also consider having producer clusters create additional datashares for you with the granularity required. You can create as many schema references to the database created from the datashare.

7. Query data in the shared objects in the datashares.

Users and groups with permissions on consumer databases and schemas on consumer clusters can explore and navigate the metadata of any shared objects. They can also explore and navigate local objects in a consumer cluster. To do this, they use JDBC or ODBC drivers or SVV\_ALL and SVV\_REDSHIFT views.

Producer clusters might have many schemas in the database, tables, and views within each schema. The users on the consumer side can see only the subset of objects that are made available through the datashare. These users can't see the entire metadata from the producer cluster. This approach helps provide granular metadata security control with data sharing.

You continue to connect to local cluster databases. But now, you can also read from the databases and schemas that are created from the datashare using the three-part database.schema.table notation. You can perform queries that span across any and all databases that are visible to you. These can be local databases on the cluster or databases created from the datashares. Consumer clusters can't connect to the databases created from the datashares.

You can access the data using full qualification. For more information, see [Examples of using a cross-database query \(p. 312\)](#).

```
SELECT * FROM sales_db.public.ticket_sales_redshift ORDER BY 1,2 LIMIT 5;

salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
commission | saletime
-----+-----+-----+-----+-----+-----+-----+
-----+-----+
 1 | 1 | 36861 | 21191 | 7872 | 1875 | 4 | 728.00 |
109.20 | 2008-02-18 02:36:48
 2 | 4 | 8117 | 11498 | 4337 | 1983 | 2 | 76.00 |
11.40 | 2008-06-06 05:00:16
 3 | 5 | 1616 | 17433 | 8647 | 1983 | 2 | 350.00 |
52.50 | 2008-06-06 08:26:17
 4 | 5 | 1616 | 19715 | 8647 | 1986 | 1 | 175.00 |
26.25 | 2008-06-09 08:38:52
 5 | 6 | 47402 | 14115 | 8240 | 2069 | 2 | 154.00 |
23.10 | 2008-08-31 09:17:02
```

You can only use SELECT statements on shared objects. However, you can create tables in the consumer cluster by querying the data from the shared objects in a different local database.

In addition to queries, consumers can create views on shared objects. Only late-binding views or materialized views are supported. Amazon Redshift doesn't support regular views on shared data. Views that consumers create can span across multiple local databases or databases created from datashares. For more information, see [CREATE VIEW \(p. 851\)](#).

```
// Connect to a local cluster database
```

```
// Create a view on shared objects and access it.
CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.ticket_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

## Working with AWS Lake Formation-managed datashares

Sharing data to AWS Lake Formation lets you centrally define AWS Lake Formation permissions of Amazon Redshift datashares and restrict user access to objects within a datashare.

### Working with Lake Formation-managed datashares as a producer

As a producer cluster administrator, follow these steps to share datashares to Lake Formation:

1. Create datashares in your cluster and authorize AWS Lake Formation to access the datashares.

Only cluster superuser and database owners can create datashares. Each datashare is associated with a database during creation. Only objects from that database can be shared in that datashare. Multiple datashares can be created on the same database with the same or different granularity of objects. There is no limit on the number of datashares you can create on a cluster.

```
CREATE DATASHARE salesshare;
```

2. Add objects to the datashare. The producer cluster administrator continues to manage datashare objects that are available. To add objects to a datashare, add the schema before adding objects. When you add a schema, Amazon Redshift doesn't add all the objects under it. You must add them explicitly. For more information, see [ALTER DATASHARE](#).

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare ADD TABLE public.ticket_sales_redshift;
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

You can also add views to a datashare. Supported views are standard views, late binding views, and materialized views.

```
CREATE VIEW public.sales_data_summary_view AS SELECT * FROM
public.ticket_sales_redshift;
ALTER DATASHARE salesshare ADD TABLE public.ticket_sales_redshift;
```

Use `ALTER DATASHARE` to share schemas, tables, and views, in a given schema. Superusers, datashare owners, or users who have `ALTER` or `ALL` permissions on the datashare can alter the datashare to add objects to or remove objects from it. Database users should be the owners of the objects or have `SELECT`, `USAGE`, or `ALL` permissions on the objects.

Use the `INCLUDENEW` clause to add any new tables and views created in a specified schema to the datashare. Only superusers can change this property for each datashare-schema pair.

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare SET INCLUDENEW = TRUE FOR SCHEMA PUBLIC;
```

3. Grant access of the datashare to a Lake Formation administrator account.

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '012345678910' VIA DATA CATALOG;
```

To revoke usage, use the following command.

```
REVOKE USAGE ON DATASHARE salesshare FROM ACCOUNT '012345678910' VIA DATA CATALOG;
```

4. Authorize access to the datashare for Lake Formation by using the `aws redshift authorize-data-share` API operation. Doing so lets Lake Formation recognize the datashare in the service account and manage associating consumers to the datashare.

```
aws redshift authorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier {"DataCatalog/<consumer-account-id>"}
```

To remove authorization from Lake Formation-managed datashares, use the `aws redshift deauthorize-data-share` API operation. By doing so, you allow AWS Lake Formation to recognize the datashare in the service account and remove authorization.

```
aws redshift deauthorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier {"DataCatalog/<consumer-account-id>"}
```

At any time, if the producer cluster administrator decides that there is no longer a need to share data with the consumer cluster, they can use `DROP DATASHARE` to delete the datashare, deauthorize the datashare, or revoke datashare permissions. The associated permissions and objects in Lake Formation are not automatically deleted.

```
DROP DATASHARE salesshare;
```

After authorizing the Lake Formation account to manage the datashare, the Lake Formation administrator can discover the shared datashare and create a database in the AWS Glue Data Catalog linking to the datashare. The Lake Formation administrator must also create local resources that define how objects within the datashare should map to objects within Lake Formation. For more information about discovering datashares and creating local resources, see [Managing permissions for data in an Amazon Redshift datashare](#).

## Working with Lake Formation-managed datashares as a consumer

After the AWS Lake Formation administrator discovers the datashare invitation and creates a database in the AWS Glue Data Catalog that links to the datashare, the consumer cluster administrator can associate the cluster with the datashare and the database in the AWS Glue Data Catalog, create a database local to the consumer cluster, and grant access to users and groups in the Amazon Redshift consumer cluster to start querying. Follow these steps to set up querying permissions.

1. On the Amazon Redshift console, create an Redshift cluster to serve as the consumer cluster, if needed. For information on how to create a cluster, see [Creating a cluster](#).
2. To list which databases in the AWS Glue Data Catalog consumer cluster users have access to, run the [SHOW DATABASES](#) command.

```
SHOW DATABASES FROM DATA CATALOG [ACCOUNT <account-id>,<account-id2>] [LIKE <expression>]
```

Doing so lists the resources that are available from the Data Catalog, such as the AWS Glue database's ARN, database name, and information about the datashare.

3. Using the AWS Glue database ARN from SHOW DATABASES, create a local database in the consumer cluster. For more information, see [CREATE DATABASE](#).

```
CREATE DATABASE lf_db FROM ARN <lake-formation-database-ARN> WITH [NO] DATA CATALOG SCHEMA [<schema>];
```

4. Grant access on databases and schema references created from the datashares to users and groups in the consumer cluster as needed. For more information, see [GRANT](#) or [REVOKE](#). Note that users created from the [CREATE USER](#) command cannot access objects in datashare that have been shared to Lake Formation. Only users with access to both Redshift and Lake Formation can access datashares that have been shared with Lake Formation.

```
GRANT USAGE ON DATABASE sales_db TO IAM:Bob;
```

As a consumer cluster administrator, you can only assign permissions on the entire database created from the datashare to your users and groups. In some cases, you need fine-grained controls on a subset of database objects created from the datashare.

You can also create late-binding views on top of shared objects and use these to assign granular permissions. You can also consider having producer clusters create additional datashares for you with the granularity required. You can create as many schema references to the database created from the datashare.

5. Database users can use the views SVV\_EXTERNAL\_TABLES and SVV\_EXTERNAL\_COLUMNS to find all of the shared tables or columns within the AWS Glue database

```
SELECT * from svv_external_tables WHERE redshift_database_name = 'lf_db';
SELECT * from svv_external_columns WHERE redshift_database_name = 'lf_db';
```

6. Query data in the shared objects in the datashares.

Users and groups with permissions on consumer databases and schemas on consumer clusters can explore and navigate the metadata of any shared objects. They can also explore and navigate local objects in a consumer cluster. To do so, they can use the JDBC or ODBC drivers or the SVV\_ALL and SVV\_EXTERNAL views.

```
SELECT * FROM lf_db.schema.table;
```

You can only use SELECT statements on shared objects. However, you can create tables in the consumer cluster by querying the data from the shared objects in a different local database.

```
// Connect to a local cluster database

// Create a view on shared objects and access it.

CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.ticket_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

# Getting started data sharing using the console

## Topics

- [Connecting to a database \(p. 357\)](#)
- [Creating datashares \(p. 357\)](#)
- [Adding datashare objects to datashares \(p. 358\)](#)
- [Adding data consumers to datashares \(p. 358\)](#)
- [Viewing datashares \(p. 358\)](#)
- [Authorizing or removing authorization from datashares \(p. 359\)](#)
- [Associating datashares \(p. 359\)](#)
- [Removing association of datashare from data consumers \(p. 360\)](#)
- [Declining datashares \(p. 361\)](#)
- [Removing datashare objects from datashares \(p. 361\)](#)
- [Removing data consumers from datashares \(p. 362\)](#)
- [Editing datashares created in your account \(p. 362\)](#)
- [Creating databases from datashares \(p. 362\)](#)
- [Deleting datashares created in your account \(p. 363\)](#)
- [Creating data sets on AWS Data Exchange \(p. 363\)](#)
- [Editing AWS Data Exchange datashares \(p. 363\)](#)
- [Deleting AWS Data Exchange datashares created in your account \(p. 364\)](#)

Use the Amazon Redshift console to manage datashares created in your account or shared from other accounts. For datashare concepts, see [Data sharing concepts for Amazon Redshift \(p. 318\)](#).

- If you are a producer cluster administrator, you can create datashares, add data consumers, add datashare objects, create databases from datashares, edit datashares, or delete datashares from the **CLUSTERS** tab.

From the navigation menu, navigate the **Clusters** tab, choose a cluster from the cluster list. Then do one of the following:

- Choose the **Datashares** tab, choose a datashare from the **Datashares created in my namespace** section. Then do one of the following:
  - [Creating datashares \(p. 357\)](#)

When a datashare is created, you can add datashare objects or data consumers. For more information, see [Adding datashare objects to datashares \(p. 358\)](#) and [Adding data consumers to datashares \(p. 358\)](#).

- [Editing datashares created in your account \(p. 362\)](#)
- [Deleting datashares created in your account \(p. 363\)](#)
- Choose **Datashares** and choose a datashare from the **Datashares from other clusters** section. Then do one of the following:
  - [Creating datashares \(p. 357\)](#)
  - [Creating databases from datashares \(p. 362\)](#)
- Choose **Databases** and choose a database from the **Databases** section. Then choose **Create datashare**. For more information, see [Creating databases from datashares \(p. 362\)](#).

## Note

To view databases and objects within databases or to view datashares in the cluster, connect to a database. For more information, see [Connecting to a database \(p. 357\)](#).

## Connecting to a database

Connect to a database to view databases and objects within databases in this cluster or to view datashares.

The user credentials used to connect to a specified database must have the necessary permissions to view all datashares.

If there is no local connection, do one of the following:

- In the cluster details page, from the **Databases** tab, in the **Databases or Datashare objects** section, choose **Connect to database** to view database objects in the cluster.
- In the cluster details page, from the **Datashares** tab, do one of the following:
  - In the **Datashares from other clusters** section, choose **Connect to database** to view datashares from other clusters.
  - In the **Datashares created in my cluster** section, choose **Connect to database** to view datashares in your cluster.
- On the **Connect to database** window, do one of the following:
  - If you choose **Create a new connection**, choose **AWS Secrets Manager** to use a stored secret to authenticate access for the connection.

Or, choose **Temporary credentials** to use database credentials to authenticate access for the connection. Specify values for **Database name** and **Database user**.

Choose **Connect**.

- Choose **Use a recent connection** to connect to another database that you have the necessary permissions.

Amazon Redshift automatically makes the connection.

After database connection is established, you can start creating datashares, querying datashares, or creating databases from datashares.

## Creating datashares

As a producer cluster administrator, you can create datashares from the **Databases** or **Datashares** tabs in the cluster details page.

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Clusters**, then choose your cluster. The cluster details page appears.
3. In the cluster details page, do one of the following:
  - From the **Databases** tab, in the **Database** section, choose a database. The database details page appears.

Choose **Create datashare**. You can only create a datashare from a local database. If you haven't connected to the database, the Connect to database page appears. Follow the steps in [Connecting to a database \(p. 357\)](#) to connect to a database. If there is a recent connection, the Create datashare page appears.

- From the **Datashares** tab, in the **Datashares** section, connect to a database if you don't have a database connection.

In the **Datashares created in my cluster** section, choose **Create datashare**. The Create datashare page appears.

4. In the **Datashare information** section, choose one of the following:

- Choose **Datashare** to create datashares to share data for read purpose across different Amazon Redshift clusters or in the same AWS account or different AWS accounts.
- Choose **AWS Data Exchange datashare** to create datashares to license your data through AWS Data Exchange.

5. Specify values for **Datashare name**, **Database name**, and **Publicly accessible**.

When you change the database name, make a new database connection.

6. In the **Datashare objects** section, to add objects to a datashare, follow [Adding datashare objects to datashares \(p. 358\)](#).

7. In the **Data consumers** section, you can choose to publish to a Redshift account, or publish to the AWS Glue Data Catalog, which starts the process of sharing data via Lake Formation. Publishing your datashare to Redshift accounts means sharing your data with another Redshift account that acts as the consumer cluster.

**Note**

Once the datashare is created, you cannot edit the configuration to publish to the other option.

8. Choose **Create datashare**.

Amazon Redshift creates the datashare. After the datashare is created, you can create databases from the datashare.

## Adding datashare objects to datashares

Add one or more objects to the datashare. Datashare objects are read-only for data consumers.

You can create a datashare without adding datashare objects and add objects later.

A datashare becomes active only when you add at least one object to the datashare.

Add at least one schema to the datashare before adding other datashare objects. Add multiple schemas by choosing **Add and repeat**.

You can choose to add all existing objects of chosen object types from the specified schema or add specific individual objects from the specified schema. Choose the **Object types**.

In **Add objects**, choose either to **Add all existing objects of chosen object types from schema** or **Add specific objects from schema**.

You can choose **Add and repeat** to add the specified schemas and datashare objects and continue to add another and objects.

## Adding data consumers to datashares

You can add one or more data consumers to the datashares. Data consumers can be cluster namespaces that uniquely identified Amazon Redshift clusters or AWS accounts.

You must explicitly choose to turn off or turn on sharing your datashare to clusters with public access.

- Choose **Add cluster namespaces to the datashare**. Namespaces are globally unique identifier (GUID) for Amazon Redshift cluster.
- Choose **Add AWS accounts** to the datashare. The specified AWS accounts must have access permissions to the datashare.

## Viewing datashares

View datashares from the **DATASHARES** or **CLUSTERS** tab.

- Use the **DATASHARES** tab to list datashares in your account or from other accounts.
  - To view datashares created in your account, choose **In my account**, then choose the datashare you want to view.
  - To view datashares that are shared from other accounts, choose **From other accounts**, then choose the datashare you want to view.
- Use the **CLUSTERS** tab to list datashares in your cluster or from other clusters.

Connect to a database. For more information, see [Connecting to a database \(p. 357\)](#).

Then choose a datashare either from the **Datashares from other clusters** or **Datashares created in my cluster** section to view its details.

## Authorizing or removing authorization from datashares

As a producer cluster administrator, choose which data consumers to authorize to access datashares or to remove authorization from. Authorized data consumers receive notifications to take actions on datashares. If you are adding a cluster namespace as a data consumer, you don't have to perform authorization.

Prerequisite: To authorize or remove authorization for the datashare, there must be at least one data consumer added to the datashare.

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Datashares**. The datashare list page appears.
3. Choose **In my account**.
4. In the **Datashares in my account** section, do one of the following:
  - Choose one or more consumer clusters that you want to authorize. The Authorize data consumers page appears. Then choose **Authorize**.

If you chose **Publish to AWS Glue Data Catalog** when creating the datashare, you can only grant authorization of the datashare to a Lake Formation account.

For AWS Data Exchange datashare, you can only authorize one datashare at a time.

When you authorize an AWS Data Exchange datashare, you are sharing the datashare with the AWS Data Exchange service and allowing AWS Data Exchange to manage access to the datashare on your behalf. AWS Data Exchange allows access to consumers by adding consumer accounts as data consumers to the AWS Data Exchange datashare when they subscribe to the products. AWS Data Exchange doesn't have read access to the datashare.

- Choose one or more consumer clusters that you want to remove authorization from. Then choose **Remove authorization**.

After data consumers are authorized, they can access datashare objects and create a consumer database to query the data.

After authorization is removed, data consumers lose access to the datashare immediately.

## Associating datashares

As a consumer cluster administrator, you can associate one or more datashares that are shared from other accounts to your entire AWS account or specific cluster namespaces in your account.

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.

2. On the navigation menu, choose **Datashares**. The datashare list page appears.
3. Choose **From other accounts**.
4. In the **Datashares from other accounts** section, choose the datashare that you want to associate and choose **Associate**. When the Associate datashare page appears, choose one of the following Association types:
  - Choose **Entire AWS account** to associate all existing and future cluster namespaces across different AWS Regions in your AWS account with the datashare. Then choose **Associate**.  
  
If the datashare is published to the AWS Glue Data Catalog, you can only associate the datashare with the entire AWS account.
  - Choose **Specific AWS Regions and cluster namespaces** to associate one or more AWS Regions and specific cluster namespaces with the datashare.
    - a. Choose **Add Region** to add specific AWS Regions and cluster namespaces to the datashare. The **Add AWS Region** page appears.
    - b. Choose an **AWS Region**.
    - c. Do one of the following:
      - Choose **Add all cluster namespaces** to add all existing and future cluster namespaces in this Region to the datashare.
      - Choose **Add specific cluster namespaces** to add one or more specific cluster namespaces in this Region to the datashare.
      - Choose one or more cluster namespaces and choose **Add AWS Region**.
    - d. Choose **Associate**.

If you're associating the datashare with a Lake Formation account, go to the Lake Formation console to create a database, then define permissions over the database. For more information, see [Setting up permissions for Amazon Redshift datashares](#) in the AWS Lake Formation Developer Guide. Once you create a AWS Glue database or a federated database, you can use query editor v2 or any preferred SQL client with your consumer cluster to query the data. For more information, see [Working with Lake Formation-managed datashares as a consumer \(p. 354\)](#).

After the datashare is associated, the datashares become available.

You can also change datashare association at any time. When changing association from specific AWS Regions and cluster namespaces to the entire AWS account, Amazon Redshift overwrites the specific Region and cluster namespaces information with AWS account information. All the AWS Regions and cluster namespaces in the AWS account then have access to the datashare.

When changing association from specific cluster namespaces to all cluster namespaces in the specified AWS Region, all cluster namespaces in this Region then have access to the datashare.

## Removing association of datashare from data consumers

As a consumer cluster administrator, you can remove association of datashares from data consumers.

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Datashares**. The datashare list page appears.
3. Choose **From other accounts**.
4. In the **Datashares from other accounts** section, choose the datashare to remove association from data consumers.
5. In the **Data consumers** section, choose one or more data consumers to remove association from. Then choose **Remove association**.
6. When the Remove association page appears, choose **Remove association**.

After association is removed, data consumers will lose access to the datashare. You can change the data consumer association at any time.

## Declining datashares

As a consumer cluster administrator, you can reject any datashare whose state is [available or active](#). After you reject a datashare, consumer cluster users lose access to the datashare. Amazon Redshift doesn't return the rejected datashare if you call the `DescribeDataSharesForConsumer` API operation. If the producer cluster administrator runs the `DescribeDataSharesForProducer` API operation, they will see that the datashare was rejected. Once a datashare is rejected, the producer cluster administrator can authorize the datashare to a consumer cluster again, and the consumer cluster administrator can choose to associate their AWS account with the datashare or reject it.

If your AWS account has an association to a datashare and a pending association to a datashare that's managed by Lake Formation, rejecting the datashare association that's managed by Lake Formation also rejects the original datashare. To reject a specific association, the producer cluster administrator can remove authorization from a specified datashare. This action doesn't affect other datashares.

To reject a datashare, use the AWS console, the API operation `RejectDataShare`, or `reject-datasource` in the AWS CLI.

### To reject a datashare using the AWS console:

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. In the navigation menu, choose **Datashares**.
3. Choose **From other accounts**.
4. In the **Datashares from other accounts** section, choose the datashare you want to decline. When the **Decline datashare** page appears, choose **Decline**.

After you decline the datashares, you can't revert the change. Amazon Redshift removes the datashares from the list. To see the datashare again, the producer administrator must authorize it again.

## Removing datashare objects from datashares

You can remove one or more objects from a datashare by using the following procedure.

### To remove one or more objects from a datashare

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Clusters**, then choose your cluster. The cluster details page appears.
3. Choose **Datashares**.
4. In the **Datashares created in my account** section, choose **Connect to database**. For more information, see [Connecting to a database \(p. 357\)](#).
5. Choose the datashare you want to edit, then choose **Edit**. The datashare details page appears.
6. To remove one or more datashare objects to the datashare, do one of the following:
  - To remove schemas from the datashare, choose one or more schemas. Then choose **Remove**. Amazon Redshift removes the specified schemas and all the objects of the specified schemas from the datashare.
  - To remove tables and views from the datashare, choose one or more tables and views. Then choose **Remove**. Alternatively, choose **Remove by schema** to remove all tables and views in the specified schemas.

- To remove user-defined functions from the datashare, choose one or more user-defined functions. Then choose **Remove**. Alternatively, choose **Remove by schema** to remove all user-defined functions in the specified schemas.

## Removing data consumers from datashares

You can remove one or more data consumers from a datashare. Data consumers can be cluster namespaces that uniquely identified Amazon Redshift clusters or AWS accounts.

Choose one or more data consumers either from the cluster namespace IDs or AWS account, then choose **Remove**.

Amazon Redshift removes the specified data consumers from the datashare. They lose access to the datashare immediately.

## Editing datashares created in your account

Edit datashares created in your account using the console. Connect to a database first to see the list of datashares created in your account.

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Clusters**, then choose your cluster. The cluster details page appears.
3. Choose **Datashares**.
4. In the **Datashares created in my account** section, choose **Connect to database**. For more information, see [Connecting to a database \(p. 357\)](#).
5. Choose the datashare you want to edit, then choose **Edit**. The datashare details page appears.
6. Make any changes in the **Datashare objects** or **Data consumers** section.

### Note

If you chose to publish your datashare to the AWS Glue Data Catalog, you can't edit the configuration to publish the datashare to other Amazon Redshift accounts.

7. Choose **Save changes**.

Amazon Redshift updates your datashare with the changes.

## Creating databases from datashares

To start querying data in the datashare, create a database from a datashare. You can create only one database from a specified datashare.

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Clusters**, then choose your cluster. The cluster details page appears.
3. Choose **Datashares**. The datashare list appears.
4. In the **Datashares from other clusters** section, choose **Connect to database**. For more information, see [Connecting to a database \(p. 357\)](#).
5. Choose a datashare that you want to create databases from, then choose **Create database from datashare**. The Create database from datashare page appears.
6. In the **Database name**, specify a database name. The database name must be 1–64 alphanumeric characters (lowercase only) and it can't be a reserved word.

7. Choose **Create**.

After the database is created, you can query data in the database.

## Deleting datashares created in your account

Delete datashares created in your account using the console. Connect to a database first to see the list of datashares created in your account.

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Clusters**, then choose your cluster. The cluster details page appears.
3. Choose **Datashares**. The datashare list appears.
4. In the **Datashares created in my account** section, choose **Connect to database**. For more information, see [Connecting to a database \(p. 357\)](#).
5. Choose one or more datashares you want to delete, then choose **Delete**. The Delete datashares page appears.

Deleting a datashare shared with Lake Formation doesn't automatically remove the associated permissions in Lake Formation. To remove them, go to the Lake Formation console.

6. Type **Delete** to confirm deleting the specified datashares.
7. Choose **Delete**.

After datashares are deleted, datashare consumers lose access to the datashares.

## Creating data sets on AWS Data Exchange

Create data sets on AWS Data Exchange.

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Clusters**, then choose your cluster. The cluster details page appears.
3. Choose **Datashares**.
4. In the **Datashares created in my account** section, choose an AWS Data Exchange datashare.
5. Choose **Create data set on AWS Data Exchange**. For more information, see [Publishing a new product](#).

## Editing AWS Data Exchange datashares

Edit AWS Data Exchange datashares using the console. Connect to a database first to see the list of datashares created in your account.

For AWS Data Exchange datashares, you can't make changes to data consumers.

To edit the publicly accessible setting for AWS Data Exchange datashares, use the Query editor v2. Amazon Redshift generates a random one-time value to set the session variable to allow turning this setting off. For more information, see [ALTER DATASHARE usage notes \(p. 630\)](#).

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Clusters**, then choose your cluster. The cluster details page appears.
3. From the navigator menu, choose **Editor**, then **Query editor v2**.

4. If this is the first time you use the Query editor v2, configure your AWS account. By default, an AWS owned key is used to encrypt resources. For more information about configuring your AWS account, see [Configuring your AWS account](#) in the *Amazon Redshift Management Guide*.
5. To connect to the cluster that your AWS Data Exchange datashare is in, choose **Database** and the cluster name in the tree-view panel. If prompted, enter the connection parameters.
6. Copy the following SQL statement. The following example changes the publicly accessible setting of the salesshare datashare.

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
```

7. To run the copied SQL statement, choose **Queries** and paste the copied SQL statement in the query area. Then choose **Run**.

An error appears following:

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
ERROR: Alter of ADX-managed datashare salesshare requires session variable
datashare_break_glass_session_var to be set to value 'c670ba4db22f4b'
```

The value 'c670ba4db22f4b' is a random one-time value that Amazon Redshift generates when an unrecommended operation occurs.

8. Copy and paste the following sample statement into the query area. Then run the command. The `SET datashare_break_glass_session_var` command applies a permission to allow an unrecommended operation for an AWS Data Exchange datashare.

```
SET datashare_break_glass_session_var to 'c670ba4db22f4b';
```

9. Run the ALTER DATASHARE statement again.

```
ALTER DATASHARE salesshare;
```

Amazon Redshift updates your datashare with the changes.

## Deleting AWS Data Exchange datashares created in your account

Delete AWS Data Exchange datashares created in your account using the console. Connect to a database first to see the list of datashares created in your account.

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Clusters**, then choose your cluster. The cluster details page appears.
3. From the navigator menu, choose **Editor**, then **Query editor v2**.
4. If this is the first time you use the Query editor v2, configure your AWS account. By default, an AWS owned key is used to encrypt resources. For more information about configuring your AWS account, see [Configuring your AWS account](#) in the *Amazon Redshift Management Guide*.
5. To connect to the cluster that your AWS Data Exchange datashare is in, choose **Database** and the cluster name in the tree-view panel. If prompted, enter the connection parameters.
6. Copy the following SQL statement. The following example drops the salesshare datashare.

```
DROP DATASHARE salesshare
```

7. To run the copied SQL statement, choose **Queries** and paste the copied SQL statement in the query area. Then choose **Run**.

An error appears following:

```
ERROR: Drop of ADX-managed datashare salesshare requires session variable
datashare_break_glass_session_var to be set to value '620c871f890c49'
```

The value '620c871f890c49' is a random one-time value that Amazon Redshift generates when an unrecommended operation occurs.

8. Copy and paste the following sample statement into the query area. Then run the command. The `SET datashare_break_glass_session_var` command applies a permission to allow an unrecommended operation for an AWS Data Exchange datashare.

```
SET datashare_break_glass_session_var to '620c871f890c49';
```

9. Run the `DROP DATASHARE` statement again.

```
DROP DATASHARE salesshare;
```

After the datashare is deleted, datashare consumers lose access to the datashare.

Deleting a shared AWS Data Exchange datashare can breach data product terms in AWS Data Exchange.

## Getting started data sharing with AWS CloudFormation

You can automate data sharing setup by using an AWS CloudFormation stack, which provisions AWS resources. The CloudFormation stack described following sets up data sharing between two Amazon Redshift clusters in the same AWS account. Thus, you can get up and running quickly without having to run SQL statements to provision your resources.

The stack creates a datashare on the cluster that you designate. The datashare includes a table and sample read-only data. This data can be read by another Redshift cluster.

If instead you want to start sharing data in an AWS account by running SQL statements to set up a datashare and grant permissions, without using CloudFormation, see [Sharing data within an AWS account \(p. 333\)](#).

Before running the data sharing CloudFormation stack, you must be logged in with a user that has permission to create an IAM role and a Lambda function. You also need two Amazon Redshift clusters in the same account. You use one, the *producer*, to share the sample data, and the other, the *consumer*, to read it. The primary requirement for these clusters is that each use RA3 nodes. For additional requirements, see [Considerations when using data sharing in Amazon Redshift \(p. 320\)](#).

For more information about getting started setting up an Amazon Redshift cluster, see [Getting started with Amazon Redshift](#). For more information about automating setup with CloudFormation, see [What is AWS CloudFormation?](#)

Use the following procedure to launch your CloudFormation stack for Amazon Redshift data sharing. Before doing so, make sure you have two Amazon Redshift clusters in the same account.

### To launch your CloudFormation stack for Amazon Redshift data sharing:

1. Click [Launch CFN stack](#), which takes you to the CloudFormation service in the AWS Management Console.

If you are prompted, sign in.

The stack creation process starts, referencing a CloudFormation template file, which is stored in Amazon S3. A CloudFormation *template* is a text file in JSON format that declares AWS resources that make up a stack.

2. Choose **Next** to enter the stack details.
3. Under **Parameters**, for each cluster, enter the following:
  - The Amazon Redshift cluster name, for example **ra3-consumer-cluster**
  - A specific database name, for example **dev**
  - The name of a database user, for example **consumeruser**

We recommend using test clusters, because the stack creates several database objects.

Choose **Next**.

The stack options appear.

4. Choose **Next** to accept the default settings.
5. Under **Capabilities**, choose **I acknowledge that AWS CloudFormation might create IAM resources**.
6. Choose **Create stack**.

CloudFormation takes about ten minutes to build the Amazon Redshift stack using the template, creating a datashare called `myproducer_share`. The stack creates the datashare in the database specified in the stack details. Only objects from that database can be shared.

If an error occurs while the stack is created, do the following:

- Make sure that you entered the correct cluster name, database name, and database user name for each Redshift cluster.
- Make sure that your cluster has RA3 nodes.

## Querying the datashare that you created

To use the following procedure, make sure that you have the required permissions for running queries on each cluster described.

### To query your datashare:

1. Connect to the producer cluster on the database entered when your CloudFormation stack was created, using a client tool such as the Redshift query editor.
2. Query for datashares.

```
SHOW DATASHARES;
```

The preceding command returns the name of the datashare created by the stack, called `myproducer_share`. It also returns the name of the database associated with the datashare, `myconsumer_db`.

Copy the producer namespace identifier to use in a later step.

3. Describe objects in the datashare.

```
DESC DATASHARE myproducer_share;
```

When you describe the datashare, it returns properties for tables and views. The stack adds tables and views with sample data to the producer database, for example `tickit_sales` and `tickit_sales_view`. For more information about the Redshift sample data, see [Sample database](#).

You don't have to delegate permissions on the datashare to run queries. The stack grants the necessary permissions.

4. Connect to the consumer cluster using your client tool. Describe the datashare, specifying the producer's namespace.

```
DESC DATASHARE myproducer_share OF NAMESPACE '<namespace id>'; --specify the unique identifier for the producer namespace
```

5. You can query tables in the datashare by specifying the datashare's database and schema.

```
SELECT * FROM myconsumer_db.myproducer_schema.tickit_sales_view;
```

The query runs against the view in the shared schema. You can't connect directly to databases created from datashares. They are read-only.

6. Run a query that includes aggregations.

```
SELECT * FROM myconsumer_db.myproducer_schema.tickit_sales ORDER BY 1,2 LIMIT 5;
```

The query returns sales and seller data from the sample Tickit data.

For more examples of datashare queries, see [Sharing data within an AWS account \(p. 333\)](#).

# Ingesting and querying semistructured data in Amazon Redshift

By using *semistructured data support* in Amazon Redshift, you can ingest and store semistructured data in your Amazon Redshift data warehouses. Using the SUPER data type and PartiQL language, Amazon Redshift expands data warehouse capability to integrate with both SQL and NoSQL data sources. This way, Amazon Redshift enables efficient analytics on relational and semistructured stored data such as JSON.

Amazon Redshift offers two forms of semistructured data support: the SUPER data type and Amazon Redshift Spectrum.

Use the SUPER data type if you need to insert or update small batches of JSON data with low latency. Also, use SUPER when your query requires strong consistency, predictable query performance, complex query support, and ease of use with evolving schemas and schemaless data.

In contrast, use Amazon Redshift Spectrum with an open file format if your data query requires integration with other AWS services and with data mainly stored in Amazon S3 for archival purposes.

## Use cases for the SUPER data type

Semistructured data support using the SUPER data type in Amazon Redshift provides superior performance, flexibility, and ease of use. The following use cases help demonstrate how you can use semistructured data support with SUPER.

**Rapid and flexible insertion of JSON data** – Amazon Redshift supports rapid transactions that can parse JSON and store it as a SUPER value. The insert transactions can operate up to five times faster than performing the same insertions into tables that have shredded the attributes of SUPER into conventional columns. For example, suppose that the incoming JSON is of the form {"a":..., "b":..., "c":..., ...}. You can accelerate the insert performance many times by storing the incoming JSON into a table TJ with a single SUPER column S, instead of storing it into a conventional table TR with columns "a", "b", "c", and so on. When there are hundreds of attributes in the JSON, the performance advantage of SUPER data type becomes substantial.

Also, SUPER data type doesn't need a regular schema. You don't need to introspect and clean up the incoming JSON before storing it. For example, suppose an incoming JSON has a string "c" attribute and others that have an integer "c" attribute, without the SUPER data type. In this case, you have to either separate c\_string and c\_int columns or clean up the data. In contrast, with the SUPER data type, all JSON data is stored during ingestion without the loss of information. Later, you can use the PartiQL extension of SQL to analyze the information.

**Flexible queries for discovery** – After you have stored your semistructured data (such as JSON) into a SUPER data value, you can query it without imposing a schema. You can use PartiQL dynamic typing and lax semantics to run your queries and discover the deeply nested data you need, without the need to impose a schema before query.

**Flexible queries for extract, load, transform (ETL) operations into conventional materialized views**

– After you have stored your schemaless and semistructured data into SUPER, you can use PartiQL materialized views to introspect the data and shred them into materialized views.

The materialized views with the shredded data are a good example of performance and usability advantages to your classic analytics cases. When you perform analytics on the shredded data, the columnar organization of Amazon Redshift materialized views provides better performance. Furthermore, users and business intelligence (BI) tools that require a conventional schema for ingested data can use views (either materialized or virtual) as the conventional schema presentation of the data.

After your PartiQL materialized views have extracted the data found in JSON or SUPER into conventional columnar materialized views, you can query the materialized views. For more information on how the SUPER data type works with materialized views, see [Using SUPER data type with materialized views \(p. 391\)](#).

For information about the SUPER data type, see [SUPER type \(p. 594\)](#).

## Concepts for SUPER data type use

Following, you can find some Amazon Redshift SUPER data type concepts.

**Understand what the SUPER data type is in Amazon Redshift** – The SUPER data type is an Amazon Redshift data type that enables the storage of schemaless arrays and structures that contain Amazon Redshift scalars and possibly nested arrays and structures. The SUPER data type can natively store different formats of semistructured data, such as JSON or data originating from document-oriented sources. You can add a new SUPER column to store semistructured data and write queries that access the SUPER column, along with the usual scalar columns. For more information about the SUPER data type, see [SUPER type \(p. 594\)](#).

**Ingest schemaless JSON into SUPER** – With the flexible semistructured SUPER data type, Amazon Redshift can receive and ingest schemaless JSON into a SUPER value. For example, Amazon Redshift can ingest the JSON value [10.5, "first"] into a SUPER value [10.5, 'first'], that is an array containing the Amazon Redshift decimal 10.5 and varchar 'first'. Amazon Redshift can ingest the JSON into a SUPER value using the COPY command or the JSON parse function, such as `json_parse('[10.5, "first"]')`. Both COPY and json\_parse ingest JSON using strict parsing semantics by default. You can also construct SUPER values including arrays and structures, using the database data themselves.

The SUPER column requires no schema modifications while ingesting the irregular structures of schemaless JSON. For example, while analyzing a click-stream, you initially store in the SUPER column "click" structures with attributes "IP" and "time". You can add an attribute "customer id" without changing your schema in order to ingest such changes.

The native format used for the SUPER data type is a binary format that requires lesser space than the JSON value in its textual form. This enables faster ingestion and runtime processing of SUPER values at query.

**Query SUPER data with PartiQL** – PartiQL is a backward-compatible extension of SQL-92 that many AWS services currently use. With the use of PartiQL, familiar SQL constructs seamlessly combine access to both the classic, tabular SQL data and the semistructured data of SUPER. You can perform object and array navigation and unnest arrays. PartiQL extends the standard SQL language to declaratively express and process nested and multivalued data.

PartiQL is an extension of SQL where the nested and schemaless data of SUPER columns are first-class citizens. PartiQL doesn't require all query expressions to be type-checked during query compilation time. This approach enables query expressions that contain the SUPER data type to be dynamically typed during query execution when the actual types of the data inside the SUPER columns are accessed. Also, PartiQL operates in a lax mode where type inconsistencies don't cause failures but return null. The combination of schemaless and lax query processing makes PartiQL ideal for extract, load, transfer (ELT) applications where your SQL query evaluates the JSON data that are ingested in the SUPER columns.

**Integrate with Redshift Spectrum** – Amazon Redshift supports multiple aspects of PartiQL when running Redshift Spectrum queries over JSON, Parquet, and other formats that have nested data.

Redshift Spectrum only supports nested data that has schemas. For example, with Redshift Spectrum you can declare that your JSON data have an attribute *nested\_schemaful\_example* in a schema `ARRAY<STRUCT<a:INTEGER, b:DECIMAL(5,2)>>`. The schema of this attribute determines that the data always contains an array, which contains a structure with integer *a* and decimal *b*. If the data changes to include more attributes, the type also changes. In contrast, the SUPER data type requires no schema. You can store arrays with structure elements that have different attributes or types. Also, some values can be stored outside arrays.

For information about functions that support the SUPER data type, see the following:

- [ABS function \(p. 1156\)](#)
- [CEILING \(or CEIL\) function \(p. 1160\)](#)
- [FLOOR function \(p. 1164\)](#)
- [ROUND function \(p. 1172\)](#)
- [SIGN function \(p. 1174\)](#)
- [TRUNC function \(p. 1177\)](#)

## Considerations for SUPER data

When working with SUPER data, consider the following:

- Use JDBC driver version 1.2.50, ODBC driver version 1.4.17 or later, and Amazon Redshift Python driver version 2.0.872 or later.

For information about JDBC drivers, see [Configuring a JDBC connection](#).

For information about ODBC drivers, see [Configuring an ODBC connection](#).

- Find the schema examples used in the following topics at [SUPER sample dataset \(p. 370\)](#).
- All the SQL code examples used in the following topics are included with the same S3 prefix for download. These include the data definition language (DDL) and COPY statements, and also certain TPC-H modified queries that work with SUPER.

To view or download the SQL files, do one of the following:

- Download the [SUPER tutorial SQL file](#) and [TPC-H file](#).
- Using the Amazon S3 CLI, run the following command. You can use your own target path.

```
aws s3 cp s3://redshift-downloads/semistructured/tutorialscripts/semistructured-tutorial.sql /target/path
aws s3 cp s3://redshift-downloads/semistructured/tutorialscripts/super_tpch_queries.sql /target/path
```

For more information about SUPER configurations, see [SUPER configurations \(p. 386\)](#).

## SUPER sample dataset

The table schema and data model used for ingestion and query examples are defined as follows.

```
/*customer-orders-lineitem*/
CREATE TABLE customer_orders_lineitem
(c_custkey bigint
,c_name varchar
,c_address varchar
```

```

,c_nationkey smallint
,c_phone varchar
,c_acctbal decimal(12,2)
,c_mktsegment varchar
,c_comment varchar
,c_orders super
);

/* Datamodel of documents to be stored in c_orders Super column would be as follows*/
ARRAY < STRUCT < o_orderkey:bigint
 ,o_orderstatus:string
 ,o_totalprice:double
 ,o_orderdate:string
 ,o_orderpriority:string
 ,o_clerk:string
 ,o_shippriority:int
 ,o_comment:string
 ,o_lineitems:ARRAY < STRUCT < l_partkey:bigint
 ,l_suppkey:bigint
 ,l_linenumber:int
 ,l_quantity:double
 ,l_extendedprice:double
 ,l_discount:double
 ,l_tax:double
 ,l_returnflag:string
 ,l_linestatus:string
 ,l_shipdate:string
 ,l_commitdate:string
 ,l_receiptdate:string
 ,l_shipinstruct:string
 ,l_shipmode:string
 ,l_comment:string
 > >
 > >

/*part*/
CREATE TABLE part
(
 p_partkey bigint
 ,p_name varchar
 ,p_mfgr varchar
 ,p_brand varchar
 ,p_type varchar
 ,p_size int
 ,p_container varchar
 ,p_retailprice decimal(12,2)
 ,p_comment varchar
);

/*region-nations*/
CREATE TABLE region_nations
(
 r_regionkey smallint
 ,r_name varchar
 ,r_comment varchar
 ,r_nations super
);

/* Datamodel of documents to be stored in r_nations Super column would be as follows*/
ARRAY < STRUCT < n_nationkey:int,n_name:string,n_comment:string > >

/*supplier-partsupp*/
CREATE TABLE supplier_partsupp
(
 s_suppkey bigint
 ,s_name varchar

```

```

,s_address varchar
,s_nationkey smallint
,s_phone varchar
,s_acctbal double precision
,s_comment varchar
,s_partsupps super
);

/* Datamodel of documents to be stored in s_partsupps Super column would be as follows*/
ARRAY < STRUCT < ps_partkey:bigint,ps_availqty:int,ps_supplycost:double,ps_comment:string
> >

```

## Loading semistructured data into Amazon Redshift

Use the SUPER data type to persist and query hierarchical and generic data in Amazon Redshift. Amazon Redshift introduces the `json_parse` function to parse data in JSON format and convert it into the SUPER representation. Amazon Redshift also supports loading SUPER columns using the `COPY` command. The supported file formats are JSON, Avro, text, comma-separated value (CSV) format, Parquet, and ORC.

For information about the `json_parse` function, see [JSON\\_PARSE function \(p. 1149\)](#).

The default encoding for SUPER data type is ZSTD.

## Parsing JSON documents to SUPER columns

You can insert or update JSON data into a SUPER column using the `json_parse` function. The function parses data in JSON format and converts it into the SUPER data type, which you can use in `INSERT` or `UPDATE` statements.

The following example inserts JSON data into a SUPER column. If the `json_parse` function is missing in the query, Amazon Redshift treats the value as a single string instead of a JSON-formatted string that must be parsed.

If you update a SUPER data column, Amazon Redshift requires the complete document to be passed to column values. Amazon Redshift doesn't support partial update.

```

INSERT INTO region_nations VALUES(),
 'lar deposits. blithely final packages cajole. regular waters are final requests.
regular accounts are according to',
 'AFRICA',
 JSON_PARSE('{"r_nations":[
 {"n_comment":" haggle. carefully final deposits detect slyly agai",
 "n_nationkey":0,
 "n_name":"ALGERIA"
 },
 {"n_comment":"ven packages wake quickly. regu",
 "n_nationkey":5,
 "n_name":"ETHIOPIA"
 },
 {"n_comment":" pending excuses haggle furiously deposits. pending, express pinto
beans wake fluffily past t",
 "n_nationkey":14,
 "n_name":"KENYA"
 },
 {"n_comment":"rns. blithely bold courts among the closely regular packages use
furiously bold platelets?",
 "n_nationkey":15,
 "n_name":"MOROCCO"
 }
]}')

```

```
 },
 {"n_comment":"s. ironic, unusual asymptotes wake blithely r",
 "n_nationkey":16,
 "n_name":"MOZAMBIQUE"
 }
]
});
```

## Using COPY to load SUPER columns in Amazon Redshift

In the following sections, you can learn about different ways to use the COPY command to load JSON data into Amazon Redshift.

### Copying data from JSON and Avro

By using semistructured data support in Amazon Redshift, you can load a JSON document without shredding the attributes of its JSON structures into multiple columns.

Amazon Redshift provides two methods to ingest JSON document using COPY, even with a JSON structure that is fully or partially unknown:

1. Store the data deriving from a JSON document into a single SUPER data column using the `noshred` option. This method is useful when the schema isn't known or is expected to change. Thus, this method makes it easier to store the entire tuple in a single SUPER column.
2. Shred the JSON document into multiple Amazon Redshift columns using the `auto` or `jsonpaths` option. Attributes can be Amazon Redshift scalars or SUPER values.

You can use these options with the JSON or Avro formats.

The maximum size for a JSON object before shredding is 4 MB.

### Copying a JSON document into a single SUPER data column

To copy a JSON document into a single SUPER data column, create a table with a single SUPER data column.

```
CREATE TABLE region_nations_noshred (rdata SUPER);
```

Copy the data from Amazon S3 into the single SUPER data column. To ingest the JSON source data into a single SUPER data column, specify the `noshred` option in the `FORMAT JSON` clause.

```
COPY region_nations_noshred FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 'noshred';
```

After COPY has successfully ingested the JSON, your table has a `rdata` SUPER data column that contains the data of the entire JSON object. The ingested data maintains all the properties of the JSON hierarchy. However, the leaves are converted to Amazon Redshift scalar types for efficient query processing.

Use the following query to retrieve the original JSON string.

```
SELECT rdata FROM region_nations_noshred;
```

When Amazon Redshift generates a SUPER data column, it becomes accessible using JDBC as a string through JSON serialization. For more information, see [Serializing complex nested JSON \(p. 302\)](#).

## Copying a JSON document into multiple SUPER data columns

You can shred a JSON document into multiple columns that can be either SUPER data columns or Amazon Redshift scalar types. Amazon Redshift spreads different portions of the JSON object to different columns.

```
CREATE TABLE region_nations
(
 r_regionkey smallint
 ,r_name varchar
 ,r_comment varchar
 ,r_nations super
);
```

To copy the data of the previous example into the table, specify the AUTO option in the FORMAT JSON clause to split the JSON value across multiple columns. COPY matches the top-level JSON attributes with column names and allows nested values to be ingested as SUPER values, such as JSON arrays and objects.

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 'auto';
```

When the JSON attribute names are in mixed upper and lower cases, specify the auto ignorecase option in the FORMAT JSON clause. For more information about the COPY command, see [Load from JSON data using the 'auto ignorecase' option \(p. 735\)](#).

In some cases, there is a mismatch between column names and JSON attributes or the attribute to load is nested more than a level deep. If so, use a jsonpaths file to manually map JSON attributes to Amazon Redshift columns.

```
CREATE TABLE nations
(
 regionkey smallint
 ,name varchar
 ,comment super
 ,nations super
);
```

Suppose that you want to load data to a table where the column names don't match the JSON attributes. In the following example, the nations table is such a table. You can create a jsonpaths file that maps the paths of attributes to the table columns by their position in the jsonpaths array.

```
{"jsonpaths": [
 "$.r_regionkey",
 "$.r_name",
 "$.r_comment",
 "$.r_nations
]
}
```

The location of the jsonpaths file is used as the argument to FORMAT JSON.

```
COPY nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/
region_nation'
```

```
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 's3://redshift-downloads/semistructured/tpch-nested/data/jsonpaths/
nations_jsonpaths.json';
```

Use the following query to access the table that shows data spread to multiple columns. The SUPER data columns are printed using the JSON format.

```
SELECT r_regionkey,r_name,r_comment,r_nations[0].n_nationkey FROM region_nations ORDER BY
1,2,3 LIMIT 1;
```

Jsonpaths files map fields in the JSON document to table columns. You can extract additional columns, such as distribution and sort keys, while still loading the complete document as a SUPER column. The following query loads the complete document to the nations column. The name column is the sort key and the regionkey column is the distribution key.

```
CREATE TABLE nations_sorted (
 regionkey smallint,
 name varchar,
 nations super
) DISTKEY(regionkey) SORTKEY(name);
```

The root jsonpath "\$" maps to the root of the document as follows:

```
{"jsonpaths": [
 "$.r_regionkey",
 "$.r_name",
 "$"
]}
```

The location of the jsonpaths file is used as the argument to FORMAT JSON.

```
COPY nations_sorted FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 's3://redshift-downloads/semistructured/tpch-nested/data/jsonpaths/
nations_sorted_jsonpaths.json';
```

## Copying data from text and CSV

Amazon Redshift represents SUPER columns in text and CSV formats as serialized JSON. Valid JSON formatting is required for SUPER columns to load with the correct type information. Unquote objects, arrays, numbers, booleans, and null values. Wrap string values in double quotes. SUPER columns use standard escaping rules for text and CSV formats. For CSV, delimiters are escaped according to the CSV standard. For text, if the chosen delimiter might also appear in a SUPER field, use the ESCAPE option during COPY and UNLOAD.

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/csv/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT CSV;
```

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/text/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
DELIMITER ','
ESCAPE;
```

## Copying data from columnar-format Parquet and ORC

If your semistructured or nested data is already available in either Apache Parquet or Apache ORC format, you can use the COPY command to ingest data into Amazon Redshift.

The Amazon Redshift table structure should match the number of columns and the column data types of the Parquet or ORC files. By specifying SERIALIZEJSON in the COPY command, you can load any column type in the file that aligns with a SUPER column in the table as SUPER. This includes structure and array types.

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/parquet/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT PARQUET SERIALIZEJSON;
```

The following example uses an ORC format.

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/orc/
region_nation'
IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT ORC SERIALIZEJSON;
```

When the attributes of the date or time data types are in ORC, Amazon Redshift converts them to varchar upon encoding them in SUPER.

## Unloading semistructured data

You can unload tables with SUPER data columns to Amazon S3 in different formats.

### Topics

- [Unloading semistructured data in CSV or text formats \(p. 376\)](#)
- [Unloading semistructured data in the Parquet format \(p. 377\)](#)

## Unloading semistructured data in CSV or text formats

You can unload tables with SUPER data columns to Amazon S3 in a comma-separated value (CSV) or text format. Using a combination of navigation and unnest clauses, Amazon Redshift unloads hierarchical data in SUPER data format to Amazon S3 in CSV or text formats. Subsequently, you can create external tables against unloaded data and query them using Redshift Spectrum. For information on using UNLOAD and the required IAM permissions, see [UNLOAD \(p. 991\)](#).

The following example unloads data into Amazon S3.

```
UNLOAD ('SELECT * FROM region_nations')
TO 's3://xxxxxx/'
IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3-Write'
DELIMITER AS '|'
GZIP
ALLOWOVERWRITE;
```

Unlike other data types where a user-defined string represents a null value, Amazon Redshift exports the SUPER data columns using the JSON format and represents it as *null* as determined by the JSON format. As a result, SUPER data columns ignore the NULL [AS] option used in UNLOAD commands.

## Unloading semistructured data in the Parquet format

You can unload tables with SUPER data columns to Amazon S3 in the Parquet format. Amazon Redshift represents SUPER columns in Parquet as the JSON data type. This enables semistructured data to be represented in Parquet. You can query these columns using Redshift Spectrum or ingest them back to Amazon Redshift using the COPY command. For information on using UNLOAD and the required IAM permissions, see [UNLOAD \(p. 991\)](#).

The following example unloads data into Amazon S3 in the Parquet format.

```
UNLOAD ('SELECT * FROM region_nations')
TO 's3://xxxxxx/'
IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3-Write'
FORMAT PARQUET;
```

## Querying semistructured data

Amazon Redshift uses the PartiQL language to offer SQL-compatible access to relational, semistructured, and nested data.

PartiQL operates with dynamic types. This approach enables intuitive filtering, joining, and aggregation on the combination of structured, semistructured, and nested datasets. The PartiQL syntax uses dotted notation and array subscript for path navigation when accessing nested data. It also enables the FROM clause items to iterate over arrays and use for unnest operations. Following, you can find descriptions of the different query patterns that combine the use of the SUPER data type with path and array navigation, unnesting, unpivoting, and joins.

## Navigation

Amazon Redshift uses PartiQL to enable navigation into arrays and structures using the [...] bracket and dot notation respectively. Furthermore, you can mix navigation into structures using the dot notation and arrays using the bracket notation. For example, the following example assumes that the `c_orders` SUPER data column is an array with a structure and an attribute is named `o_orderkey`.

To ingest data in the `customer_orders_lineitem` table, run the following command. Replace the IAM role with your own credentials.

```
COPY customer_orders_lineitem FROM 's3://redshift-downloads/semistructured/tpch-nested/
data/json/customer_orders_lineitem'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 'auto';

SELECT c_orders[0].o_orderkey FROM customer_orders_lineitem;
```

Amazon Redshift also uses a table alias as a prefix to the notation. The following example is the same query as the previous example.

```
SELECT cust.c_orders[0].o_orderkey FROM customer_orders_lineitem AS cust;
```

You can use the dot and bracket notations in all types of queries, such as filtering, join, and aggregation. You can use these notations in a query in which there are normally column references. The following example uses a SELECT statement that filters results.

```
SELECT count(*) FROM customer_orders_lineitem WHERE c_orders[0]. o_orderkey IS NOT NULL;
```

The following example uses the bracket and dot navigation in both GROUP BY and ORDER BY clauses.

```
SELECT c_orders[0].o_orderdate,
 c_orders[0].o_orderstatus,
 count(*)
 FROM customer_orders_lineitem
 WHERE c_orders[0].o_orderkey IS NOT NULL
 GROUP BY c_orders[0].o_orderstatus,
 c_orders[0].o_orderdate
 ORDER BY c_orders[0].o_orderdate;
```

## Unnesting queries

To unnest queries, Amazon Redshift uses the PartiQL syntax to iterate over SUPER arrays. It does this by navigating the array using the FROM clause of a query. Using the previous example, the following example iterates over the attribute values for `c_orders`.

```
SELECT c.* , o
 FROM customer_orders_lineitem c, c.c_orders o;
```

The unnesting syntax is an extension of the FROM clause. In standard SQL, the FROM clause `x (AS) y` means that `y` iterates over each tuple in relation `x`. In this case, `x` refers to a relation and `y` refers to an alias for relation `x`. Similarly, the PartiQL syntax of unnesting using the FROM clause item `x (AS) y` means that `y` iterates over each (SUPER) value in (SUPER) array expression `x`. In this case, `x` is a SUPER expression and `y` is an alias for `x`.

The left operand can also use the dot and bracket notation for regular navigation. In the previous example, `customer_orders_lineitem c` is the iteration over the `customer_order_lineitem` base table and `c.c_orders o` is the iteration over the `c.c_orders` array. To iterate over the `o.lineitems` attribute, which is an array within an array, you add multiple clauses.

```
SELECT c.* , o , l
 FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

Amazon Redshift also supports an array index when iterating over the array using the AT keyword. The clause `x AS y AT z` iterates over array `x` and generates the field `z`, which is the array index. The following example shows how an array index works.

```
SELECT c_name,
 orders.o_orderkey AS orderkey,
 index AS orderkey_index
 FROM customer_orders_lineitem c, c.c_orders AS orders AT index
 ORDER BY orderkey_index;

c_name | orderkey | orderkey_index
-----+-----+-----
Customer#000008251 | 3020007 | 0
Customer#000009452 | 4043971 | 0
(2 rows)
```

The following example iterates over a scalar array.

```
CREATE TABLE bar AS SELECT json_parse('{"scalar_array": [1, 2.3, 45000000]}') AS data;

SELECT index, element
 FROM bar AS b, b.data.scalar_array AS element AT index;

index | element
-----+-----
0 | 1
1 | 2.3
```

|              |
|--------------|
| 2   45000000 |
| (3 rows)     |

The following example iterates over an array of multiple levels. The example uses multiple unnest clauses to iterate into the innermost arrays. The `f.multi_level_array AS array` iterates over `multi_level_array`. The array AS element is the iteration over the arrays within `multi_level_array`.

```
CREATE TABLE foo AS SELECT json_parse('[[[1.1, 1.2], [2.1, 2.2], [3.1, 3.2]]') AS multi_level_array;

SELECT array, element FROM foo AS f, f.multi_level_array AS array, array AS element;

array | element
-----+-----
[1.1,1.2] | 1.1
[1.1,1.2] | 1.2
[2.1,2.2] | 2.1
[2.1,2.2] | 2.2
[3.1,3.2] | 3.1
[3.1,3.2] | 3.2
(6 rows)
```

For more information about the FROM clause, see [FROM clause \(p. 937\)](#).

## Object unpivoting

To perform object unpivoting, Amazon Redshift uses the PartiQL syntax to iterate over SUPER objects. It does this using the FROM clause of a query with the UNPIVOT keyword. The following query iterates over the `c.c_orders[0]` object.

```
SELECT attr as attribute_name, json_typeof(val) as value_type
FROM customer_orders_lineitem c, UNPIVOT c.c_orders[0] AS val AT attr
WHERE c_custkey = 9451;

attribute_name | value_type
-----+-----
o_orderstatus | string
o_clerk | string
o_lineitems | array
o_orderdate | string
o_shippriority | number
o_totalprice | number
o_orderkey | number
o_comment | string
o_orderpriority | string
(9 rows)
```

As with unnesting, the unpivoting syntax is also an extension of the FROM clause. The difference is that the unpivoting syntax uses the UNPIVOT keyword to indicate that it's iterating over an object instead of an array. It uses the AS `value_alias` for iteration over all the values inside an object and uses the AT `attribute_alias` for iterating over all the attributes.

Amazon Redshift also supports using object unpivoting and array unnesting in a single FROM clause as follows.

```
SELECT attr as attribute_name, val as object_value
FROM customer_orders_lineitem c, c.c_orders AS o, UNPIVOT o AS val AT attr
WHERE c_custkey = 9451;
```

When you use object unpivoting, Amazon Redshift doesn't support correlated unpivoting. Specifically, suppose that you have a case where there are multiple examples of unpivoting in different query levels and the inner unpivoting references the outer one. Amazon Redshift doesn't support this type of multiple unpivoting.

For more information about the FROM clause, see [FROM clause \(p. 937\)](#). For examples that show how to query structured data, with PIVOT and UNPIVOT, see [PIVOT and UNPIVOT examples \(p. 940\)](#).

## Dynamic typing

Dynamic typing doesn't require explicit casting of data that is extracted from the dot and bracket paths. Amazon Redshift uses dynamic typing to process schemaless SUPER data without the need to declare the data types before you use them in your query. Dynamic typing uses the results of navigating into SUPER data columns without having to explicitly cast them into Amazon Redshift types. Dynamic typing is most useful in joins and GROUP BY clauses. The following example uses a SELECT statement that requires no explicit casting of the dot and bracket expressions to the usual Amazon Redshift types. For information about type compatibility and conversion, see [Type compatibility and conversion \(p. 597\)](#).

```
SELECT c_orders[0].o_orderkey
FROM customer_orders_lineitem
WHERE c_orders[0].o_orderstatus = 'P';
```

The equality sign in this query evaluates to true when c\_orders[0].o\_orderstatus is the string 'P'. In all other cases, the equality sign evaluates to false, including the cases where the arguments of the equality are different types.

## Dynamic and static typing

Without using dynamic typing, you can't determine whether c\_orders[0].o\_orderstatus is a string, an integer, or a structure. You can only determine that c\_orders[0].o\_orderstatus is a SUPER data type, which can be an Amazon Redshift scalar, an array, or a structure. The static type of c\_orders[0].o\_orderstatus is a SUPER data type. Conventionally, a type is implicitly a static type in SQL.

Amazon Redshift uses dynamic typing to the processing of schemaless data. When the query evaluates the data, c\_orders[0].o\_orderstatus turns out to be a specific type. For example, evaluating c\_orders[0].o\_orderstatus on the first record of customer\_orders\_lineitem may result into an integer. Evaluating on the second record may result into a string. These are the dynamic types of the expression.

When using an SQL operator or function with dot and bracket expressions that have dynamic types, Amazon Redshift produces results similar to using standard SQL operator or function with the respective static types. In this example, when the dynamic type of the path expression is a string, the comparison with the string 'P' is meaningful. Whenever the dynamic type of c\_orders[0].o\_orderstatus is any other data type except being a string, the equality returns false. Other functions return null when mistyped arguments are used.

The following example writes the previous query with static typing:

```
SELECT c_custkey
FROM customer_orders_lineitem
WHERE CASE WHEN JSON_TYPEOF(c_orders[0].o_orderstatus) = 'string'
 THEN c_orders[0].o_orderstatus::VARCHAR = 'P'
 ELSE FALSE END;
```

Note the following distinction between equality predicates and comparison predicates. In the previous example, if you replace the equality predicate with a less-than-or-equal predicate, the semantics produce null instead of false.

```
SELECT c_orders[0]. o_orderkey
```

```
FROM customer_orders_lineitem
WHERE c_orders[0].o_orderstatus <= 'P';
```

In this example, if `c_orders[0].o_orderstatus` is a string, Amazon Redshift returns true if it is alphabetically equal to or smaller than 'P'. Amazon Redshift returns false if it is alphabetically larger than 'P'. However, if `c_orders[0].o_orderstatus` is not a string, Amazon Redshift returns null since Amazon Redshift can't compare values of different types, as shown in the following query:

```
SELECT c_custkey
FROM customer_orders_lineitem
WHERE CASE WHEN JSON_TYPEOF(c_orders[0].o_orderstatus) = 'string'
THEN c_orders[0].o_orderstatus::VARCHAR <= 'P'
ELSE NULL END;
```

Dynamic typing doesn't exclude from comparisons of types that are minimally comparable. For example, you can convert both CHAR and VARCHAR Amazon Redshift scalar types to SUPER. They are comparable as strings, including ignoring trailing white-space characters similar to Amazon Redshift CHAR and VARCHAR types. Similarly, integers, decimals, and floating-point values are comparable as SUPER values. Specifically for decimal columns, each value can also have a different scale. Amazon Redshift still considers them as dynamic types.

Amazon Redshift also supports equality on objects and arrays that are evaluated as deep equal, such as evaluating deep into objects or arrays and comparing all attributes. Use deep equal with caution, because the process of performing deep equal can be time-consuming.

## Using dynamic typing for joins

For joins, dynamic typing automatically matches values with different dynamic types without performing a long CASE WHEN analysis to find out what data types may appear. For example, assume that your organization changed the format that it was using for part keys over time.

The initial integer part keys issued are replaced by string part keys, such as 'A55', and later replaced again by array part keys, such as '[X, 10] combining a string and a number. Amazon Redshift doesn't have to perform a lengthy case analysis about part keys and can use joins as shown in the following example.

```
SELECT c.c_name
 ,l.l_extendedprice
 ,l.l_discount
FROM customer_orders_lineitem c
 ,c.c_orders o
 ,o.o_lineitems l
 ,supplier_partsupp s
 ,s.s_partsupps ps
WHERE l.l_partkey = ps.ps_partkey
AND c.c_nationkey = s.s_nationkey
ORDER BY c.c_name;
```

The following example shows how complex and inefficient the same query can be without using dynamic typing:

```
SELECT c.c_name
 ,l.l_extendedprice
 ,l.l_discount
FROM customer_orders_lineitem c
 ,c.c_orders o
 ,o.o_lineitems l
 ,supplier_partsupp s
 ,s.s_partsupps ps
WHERE CASE WHEN IS_INTEGER(l.l_partkey) AND IS_INTEGER(ps.ps_partkey)
```

```

THEN l.l_partkey::integer = ps.ps_partkey::integer
WHEN IS_VARCHAR(l.l_partkey) AND IS_VARCHAR(ps.ps_partkey)
THEN l.l_partkey::varchar = ps.ps_partkey::varchar
WHEN IS_ARRAY(l.l_partkey) AND IS_ARRAY(ps.ps_partkey)
 AND IS_VARCHAR(l.l_partkey[0]) AND IS_VARCHAR(ps.ps_partkey[0])
 AND IS_INTEGER(l.l_partkey[1]) AND IS_INTEGER(ps.ps_partkey[1])
THEN l.l_partkey[0]::varchar = ps.ps_partkey[0]::varchar
 AND l.l_partkey[1]::integer = ps.ps_partkey[1]::integer
ELSE FALSE END
AND c.c_nationkey = s.s_nationkey
ORDER BY c.c_name;

```

## Lax semantics

By default, navigation operations on SUPER values return null instead of returning an error out when the navigation is invalid. Object navigation is invalid if the SUPER value is not an object or if the SUPER value is an object but doesn't contain the attribute name used in the query. For example, the following query accesses an invalid attribute name in the SUPER data column `cdata`:

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

Array navigation returns null if the SUPER value is not an array or the array index is out of bounds. The following query returns null because `c_orders[1][1]` is out of bounds.

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

Lax semantics is especially useful when using dynamic typing to cast a SUPER value. Casting a SUPER value to the wrong type returns null instead of an error if the cast is invalid. For example, the following query returns null because it can't cast the string value 'Good' of the object attribute `o_orderstatus` to `INTEGER`. Amazon Redshift returns an error for a `VARCHAR` to `INTEGER` cast but not for a SUPER cast.

```
SELECT c.c_orders.o_orderstatus::integer FROM customer_orders_lineitem c;
```

## Types of introspection

SUPER data columns support inspection functions that return the dynamic type and other type information about the SUPER value. The most common example is the `JSON_TYPEOF` scalar function that returns a `VARCHAR` with values `boolean`, `number`, `string`, `object`, `array`, or `null`, depending on the dynamic type of the SUPER value. Amazon Redshift supports the following boolean functions for SUPER data columns:

- `DECIMAL_PRECISION`
- `DECIMAL_SCALE`
- `IS_ARRAY`
- `IS_BIGINT`
- `IS_CHAR`
- `IS_DECIMAL`
- `IS_FLOAT`
- `IS_INTEGER`
- `IS_OBJECT`
- `IS_SCALAR`
- `IS_SMALLINT`
- `IS_VARCHAR`

- `JSON_TYPEOF`

All these functions return false if the input value is null. `IS_SCALAR`, `IS_OBJECT`, and `IS_ARRAY` are mutually exclusive and cover all possible values except for null.

To infer the types corresponding to the data, Amazon Redshift uses the `JSON_TYPEOF` function that returns the type of (the top level of) the SUPER value as shown in the following example:

```
SELECT JSON_TYPEOF(r_nations) FROM region_nations;
json_typeof

array
(1 row)
```

```
SELECT JSON_TYPEOF(r_nations[0].n_nationkey) FROM region_nations;
json_typeof

number
```

Amazon Redshift sees this as a single long string, similar to inserting this value into a `VARCHAR` column instead of a SUPER. Since the column is SUPER, the single string is still a valid SUPER value and the difference is noted in `JSON_TYPEOF`:

```
SELECT IS_VARCHAR(r_nations[0].n_name) FROM region_nations;
is_varchar

true
(1 row)
```

```
SELECT r_nations[4].n_name FROM region_nations
WHERE CASE WHEN IS_INTEGER(r_nations[4].n_nationkey)
 THEN r_nations[4].n_nationkey::INTEGER = 15
 ELSE false END;
```

## Order by

Amazon Redshift doesn't define SUPER comparisons among values with different dynamic types. A SUPER value that is a string is neither smaller nor larger than a SUPER value that is a number. To use `ORDER BY` clauses with SUPER columns, Amazon Redshift defines a total ordering among different types to be observed when Amazon Redshift ranks SUPER values using `ORDER BY` clauses. The order among dynamic types is boolean, number, string, array, object. The following example shows the orders of different types:

```
INSERT INTO region_nations VALUES
(100, 'name1', 'comment1', 'AWS'),
(200, 'name2', 'comment2', 1),
(300, 'name3', 'comment3', ARRAY(1, 'abc', null)),
(400, 'name4', 'comment4', -2.5),
(500, 'name5', 'comment5', 'Amazon');

SELECT r_nations FROM region_nations ORDER BY r_nations;

r_nations

-2.5
1
"Amazon"
```

```
"AWS"
[1,"abc",null]
(5 rows)
```

For more information about the ORDER BY clause, see [ORDER BY clause \(p. 969\)](#).

## Operators and functions

Amazon Redshift provides the following function support of SUPER operators and functions.

### Arithmetic operators

SUPER values support all basic arithmetic operators +, -, \*, /, % using dynamic typing. The resultant type of the operation remains as SUPER. For all operators, except for the binary operator +, the input operands must be numbers. Otherwise, Amazon Redshift returns null. The distinction between decimals and floating-point values is retained when Amazon Redshift runs these operators and the dynamic type doesn't change. However, decimal scale changes when you use multiplications and divisions. Arithmetic overflows still cause query errors, they aren't changed to null. Binary operator + performs addition if the inputs are numbers or concatenation if the inputs are string. If one operand is a string and the other operand is a number, the result is null. Unary prefix operators + and - returns null if the SUPER value is not a number as shown in the following example:

```
SELECT (c_orders[0]. o_orderkey + 0.5) * c_orders[0]. o_orderkey / 10 AS math FROM
customer_orders_lineitem;
math

1757958232200.1500
(1 row)
```

Dynamic typing allows decimal values in SUPER to have different scales. Amazon Redshift treats decimal values as if they are different static types and allows all mathematical operations. Amazon Redshift computes the resulting scale dynamically based on the scales of the operands. If one of the operands is a floating-point number, then Amazon Redshift promotes the other operand to a floating-point number and generates the result as a floating-point number.

### Arithmetic functions

Amazon Redshift supports the following arithmetic functions for SUPER columns. They return null if the input isn't a number:

- FLOOR. For more information, see [FLOOR function \(p. 1164\)](#).
- CEIL and CEILING. For more information, see [CEILING \(or CEIL\) function \(p. 1160\)](#).
- ROUND. For more information, see [ROUND function \(p. 1172\)](#).
- TRUNC. For more information, see [TRUNC function \(p. 1177\)](#).
- ABS. For more information, see [ABS function \(p. 1156\)](#).

The following example uses arithmetic functions to query data:

```
SELECT x, FL00R(x), CEIL(x), ROUND(x)
FROM (
 SELECT (c_orders[0]. o_orderkey + 0.5) * c_orders[0].o_orderkey / 10 AS x
 FROM customer_orders_lineitem
);
```

| x                  | floor         | ceil          | round         |
|--------------------|---------------|---------------|---------------|
| 1389636795898.0500 | 1389636795898 | 1389636795899 | 1389636795898 |

The ABS function retains the scale of the input decimal while FLOOR, CEIL. The ROUND eliminates the scale of the input decimal.

## Array functions

Amazon Redshift supports the following array composition and utility functions array, array\_concat, subarray, array\_flatten, get\_array\_length, and split\_to\_array.

You can construct SUPER arrays from values in Amazon Redshift data types using the ARRAY function, including other SUPER values. The following example uses the variadic function ARRAY:

```
SELECT ARRAY(1, c.c_custkey, NULL, c.c_name, 'abc') FROM customer_orders_lineitem c;
array

[1,8401,null,"Customer#000008401","","abc"]
[1,9452,null,"Customer#000009452","","abc"]
[1,9451,null,"Customer#000009451","","abc"]
[1,8251,null,"Customer#000008251","","abc"]
[1,5851,null,"Customer#000005851","","abc"]
(5 rows)
```

The following example uses array concatenation with the ARRAY\_CONCAT function:

```
SELECT ARRAY_CONCAT(JSON_PARSE('[10001,10002]),JSON_PARSE('[10003,10004]));
array_concat

[10001,10002,10003,10004]
(1 row)
```

The following example uses array manipulation with the SUBARRAY function which returns a subset of the input array.

```
SELECT SUBARRAY(ARRAY('a', 'b', 'c', 'd', 'e', 'f'), 2, 3);
subarray

["c","d","e"]
(1 row)
```

The following example merges multiple levels of arrays into a single array using ARRAY\_FLATTEN:

```
SELECT x, ARRAY_FLATTEN(x) FROM (SELECT ARRAY(1, ARRAY(2, ARRAY(3, ARRAY())))) AS x;
x | array_flatten

[1,[2,[3,[]]]] | [1,2,3]
(1 row)
```

Array functions ARRAY\_CONCAT and ARRAY\_FLATTEN use dynamic typing rules. They return a null instead of an error if the input isn't an array. The GET\_ARRAY\_LENGTH function returns the length of a SUPER array given an object or array path.

```
SELECT c_name
```

```
FROM customer_orders_lineitem
WHERE GET_ARRAY_LENGTH(c_orders) = (
 SELECT MAX(GET_ARRAY_LENGTH(c_orders))
 FROM customer_orders_lineitem
);
```

The following example splits a string to an array of strings using SPLIT\_TO\_ARRAY. The function uses a delimiter as an optional parameter. If no delimiter is absent, then the default is a comma.

```
SELECT SPLIT_TO_ARRAY('12|345|6789', '|');
split_to_array

[\"12\", \"345\", \"6789\"]
```

(1 row)

## SUPER configurations

Note the following considerations of SUPER configurations when you use Amazon Redshift SUPER data type and PartiQL.

### Lax and strict modes for SUPER

When you query SUPER data, the path expression may not match the actual SUPER data structure. If you try to access a non-existent member of an object or element of an array, Amazon Redshift returns a NULL value if your query is run in the default lax mode. If you run your query in the strict mode, Amazon Redshift returns an error. The following session parameters can be set to set the lax mode on or off.

The following example uses session parameters to enable lax mode.

```
SET navigate_super_null_on_error=ON; --default lax mode for navigation
SET cast_super_null_on_error=ON; --default lax mode for casting
SET parse_super_null_on_error=OFF; --default strict mode for ingestion
```

### Accessing JSON fields with uppercase and mixedcase field names or attributes

When your JSON attribute names are in uppercase or mixedcase, you must be able to navigate SUPER type structures in a case sensitive way. To do that, you can configure enable\_case\_sensitive\_identifier to TRUE and wrap the uppercase and mixedcase attribute names with double quotation marks. You can also configure enable\_case\_sensitive\_super\_attribute to TRUE. In this case, you can use uppercase and mixedcase attribute names in your queries without wrapping them in double quotation marks.

The following example illustrates how to set enable\_case\_sensitive\_identifier to query data.

```
SET enable_case_sensitive_identifier to TRUE;

-- Accessing JSON attribute names with uppercase and mixedcase names
SELECT json_table.data.\"ITEMS\".\"Name\",
 json_table.data.\"price\"
FROM (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;
```

```

Name | price
-----+-----
"TV" | 345
(1 row)

RESET enable_case_sensitive_identifier;

-- After resetting the above configuration, the following query accessing JSON attribute
names with uppercase and mixedcase names should return null (if in lax mode).
SELECT json_table.data."ITEMS".Name,
 json_table.data."price"
FROM
 (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

name | price
-----+-----
| 345
(1 row)

```

The following example illustrates how to set `enable_case_sensitive_super_attribute` to query data.

```

SET enable_case_sensitive_super_attribute to TRUE;
-- Accessing JSON attribute names with uppercase and mixedcase names

SELECT json_table.data.ITEMS.Name,
 json_table.data.price
FROM
 (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

name | price
-----+-----
"TV" | 345
(1 row)

RESET enable_case_sensitive_super_attribute;

-- After resetting enable_case_sensitive_super_attribute, the query now returns NULL for
ITEMS.Name (if in lax mode).

SELECT json_table.data.ITEMS.Name,
 json_table.data.price
FROM
 (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

name | price
-----+-----
| 345
(1 row)

```

## Parsing options for SUPER

When you use the `JSON_PARSE` function to parse JSON strings into SUPER values, certain restrictions apply:

- The same attribute name cannot appear in the same object, but can appear in a nested object. The `json_parse_dedup_attributes` configuration option allows `JSON_PARSE` to keep only the last occurrence of duplicate attributes instead of returning an error.
- String values cannot exceed the system max varchar size of 65535 bytes. The `json_parse_truncate_strings` configuration option allows `JSON_PARSE()` to automatically

truncate strings that are longer than this limit without returning an error. This behavior affects string values only and not attribute names.

For more information about the JSON\_PARSE function, see [JSON\\_PARSE function \(p. 1149\)](#).

The following example shows how to set the json\_parse\_dedup\_attributes configuration option to the default behavior of returning an error for duplicate attributes.

```
SET json_parse_dedup_attributes=OFF; --default behavior of returning error instead of deduplicating attributes
```

The following example shows how to set the json\_parse\_truncate\_strings configuration option for the default behavior of returning an error for strings that are longer than this limit.

```
SET json_parse_truncate_strings=OFF; --default behavior of returning error instead of truncating strings
```

## Limitations

When using the SUPER data type, consider the following limitations:

- You can't define SUPER columns as either a distribution or sort key.
- An individual SUPER field or object can hold up to 1 MB of data.
- An individual value within a SUPER object is limited to the maximum length of the corresponding Amazon Redshift type. For example, a single string value loaded to SUPER is limited to the maximum VARCHAR length of 65535 bytes.
- You can't perform partial update or transform operations on SUPER columns.
- You can't use the SUPER data type and its alias in right joins or full outer joins.
- The SUPER data type doesn't support XML as inbound or outbound serialization format.
- In the FROM clause of a subquery (that is correlated or not) that references a table variable for unnesting, the query can only refer to its parent table and not other tables.
- Casting limitations

SUPER values can be cast to and from other data types, with the following exceptions:

- Amazon Redshift doesn't differentiate integers and decimals of scale 0.
- If the scale isn't zero, SUPER data type has the same behavior as other Amazon Redshift data types, except that Amazon Redshift converts SUPER-related errors to null, as shown in the following example.

```
SELECT 5::bool;
bool

True
(1 row)

SELECT 5::decimal::bool;
ERROR: cannot cast type numeric to boolean

SELECT 5::super::bool;
bool

True
(1 row)
```

```

SELECT 5.0::bool;
ERROR: cannot cast type numeric to boolean

SELECT 5.0::super::bool;
 bool

(1 row)

```

- Amazon Redshift doesn't cast the date and time types to SUPER data type. Amazon Redshift can only cast the date and time data types from SUPER data type, as shown in the following example.

```

SELECT o.o_orderdate FROM customer_orders_lineitem c,c.c_orders o;
 order_date

"2001-09-08"
(1 row)

SELECT JSON_TYPEOF(o.o_orderdate) FROM customer_orders_lineitem c,c.c_orders o;
 json_typeof

 string
(1 row)

SELECT o.o_orderdate::date FROM customer_orders_lineitem c,c.c_orders o;
 order_date

2001-09-08
(1 row)

--date/time cannot be cast to super
SELECT '2019-09-09'::date::super;
ERROR: cannot cast type date to super

```

- Cast from non-scalar values (object and array) to string returns NULL. To properly serialize these non-scalar values, don't cast them. Instead, use `json_serialize` to cast non-scalar values. The `json_serialize` function returns a varchar. Typically, you don't need to cast non-scalar values to varchar since Amazon Redshift implicitly serializes as shown in the following first example.

```

SELECT r_nations FROM region_nations WHERE r_regionkey=300;
 r_nations

[1,"abc",null]
(1 row)

SELECT r_nations::varchar FROM region_nations WHERE r_regionkey=300;
 r_nations

(1 row)

SELECT JSON_SERIALIZE(r_nations) FROM region_nations WHERE r_regionkey=300;
 json_serialize

[1,"abc",null]
(1 row)

```

- For case-insensitive databases, Amazon Redshift doesn't support the SUPER data type. For case-insensitive columns, Amazon Redshift doesn't cast them to the SUPER type. Thus, Amazon Redshift doesn't support SUPER columns interacting with case-insensitive columns that trigger casting.
- Amazon Redshift doesn't support volatile functions, such as `RANDOM()` or `TIMEofday()`, in subqueries that unnest an outer table or a left-hand side (LHS) of IN functions with such subqueries.

*The following is prerelease documentation for SUPER data type fields larger than 1 MB, which are in preview release. The documentation and the feature are both subject to change. We recommend that you use this feature only with test clusters, and not in production environments. Public preview will end on May 22, 2023. Preview clusters and preview serverless workgroups and namespaces will be removed automatically two weeks after the end of the preview. For preview terms and conditions, see Betas and Previews in [AWS Service Terms](#).*

#### Note

You can create an Amazon Redshift cluster in **Preview** to test new features of Amazon Redshift. You can't use those features in production or move your **Preview** cluster to a production cluster or a cluster on another track. For preview terms and conditions, see *Beta and Previews* in [AWS Service Terms](#).

#### To create a cluster in Preview

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Provisioned clusters dashboard**, and choose **Clusters**. The clusters for your account in the current AWS Region are listed. A subset of properties of each cluster is displayed in columns in the list.
3. A banner displays on the **Clusters** list page that introduces preview. Choose the button **Create preview cluster** to open the create cluster page.
4. Enter properties for your cluster. Choose the **Preview track** that contains the features you want to test. We recommend entering a name for the cluster that indicates that it is on a preview track. Choose options for your cluster, including options labeled as **-preview**, for the features you want to test. For general information about creating clusters, see [Creating a cluster](#) in the *Amazon Redshift Management Guide*.
5. Choose **Create preview cluster** to create a cluster in preview.
6. When your preview cluster is available, use your SQL client to load and query data.

Your cluster must be created with the preview track named: `preview_2022`.

You can create an Amazon Redshift Serverless workgroup in **Preview** to test new features of Amazon Redshift Serverless. You can't use those features in production or move your **Preview** workgroup to a production workgroup. For preview terms and conditions, see *Beta and Previews* in [AWS Service Terms](#).

#### To create a workgroup in Preview

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Serverless dashboard**, and choose **Workgroup configuration**. The workgroups for your account in the current AWS Region are listed. A subset of properties of each workgroup is displayed in columns in the list.
3. A banner displays on the **Workgroups** list page that introduces preview. Choose the button **Create preview workgroup** to open the create workgroup page.
4. Enter properties for your workgroup. We recommend entering a name for the workgroup that indicates that it is in preview. Choose options for your workgroup, including options labeled as **-preview**, for the features you want to test. Continue through the pages to enter options for your workgroup and namespace. For general information about creating workgroups, see [Creating a workgroup with a namespace](#) in the *Amazon Redshift Management Guide*.
5. Choose **Create preview workgroup** to create a workgroup in preview.

- When your preview workgroup is available, use your SQL client to load and query data.

In preview track clusters and workgroups, an individual SUPER field or object can hold up to 16 MB of data. Amazon Redshift doesn't support the following query functionality on tables with SUPER type columns that hold SUPER objects larger than 1 MB:

- Cross-functional limitations
  - Data sharing
  - Concurrency scaling
  - Query federation
- Core workflow limitations
  - Elastic resize and cross instance restore
- Ingestion limitations
  - Parquet and JSON are the only supported file formats for using the COPY command on tables holding 16 MB SUPER type objects.
- Query limitations
  - Distinct aggregates on queries that include columns holding SUPER data
  - Python user-defined functions (UDFs) that access columns holding SUPER data
  - Recursive common table expressions (CTEs) on tables with columns holding SUPER data

## Using SUPER data type with materialized views

Amazon Redshift extends the capability of materialized views to work with the SUPER data type and PartiQL in materialized views. SQL and PartiQL queries can be precomputed using incremental materialized views. For more information about materialized views, see [Creating materialized views in Amazon Redshift \(p. 198\)](#).

Once you have stored your schemaless and semistructured data into SUPER, you can use PartiQL materialized views to introspect the data and shred them into materialized views.

### Accelerating PartiQL queries

You can use materialized views to accelerate PartiQL queries that navigate and/or unnest hierarchical data in SUPER columns. Create one or more materialized views to shred the SUPER values into multiple columns and utilize the columnar organization of Amazon Redshift analytical queries. Consequently, queries make use of the materialized views.

The materialized view essentially extracts and normalizes the nested data. The level of normalization depends on how much effort you put into turning the SUPER data into conventional columnar data.

### Shredding into SUPER columns with materialized views

The following example shows a materialized view that shreds the nested data with the resulting columns still being the SUPER data type.

```
SELECT c.c_name, o.o_orderstatus
FROM customer_orders_lineitem c, c.c_orders o;
```

The following example shows a materialized view that creates conventional Amazon Redshift scalar columns from the shredded data.

```
SELECT c.c_name, c.c_orders[0].o_totalprice
FROM customer_orders_lineitem c;
```

You can create a single materialized view super\_mv to accelerate both queries.

To answer the first query, you must materialize the attribute o\_orderstatus. You can omit the attribute c\_name because it doesn't involve nested navigation nor unnesting. You must also include in the materialized view the attribute c\_custkey of customer\_orders\_lineitem to be able to join the base table with the materialized view.

To answer the second query, you must also materialize the attribute o\_totalprice and the array index o\_idx of c\_orders. Hence, you can access the index 0 of c\_orders.

```
CREATE MATERIALIZED VIEW super_mv distkey(c_custkey) sortkey(c_custkey) AS (
 SELECT c_custkey, o.o_orderstatus, o.o_totalprice, o_idx
 FROM customer_orders_lineitem c, c.c_orders o AT o_idx
);
```

The attributes o\_orderstatus and o\_totalprice of the materialized view super\_mv are SUPER.

The materialized view super\_mv will be refreshed incrementally upon changes to the base table customer\_orders\_lineitem.

```
REFRESH MATERIALIZED VIEW super_mv;
INFO: Materialized view super_mv was incrementally updated successfully.
```

To rewrite the first PartiQL query as a regular SQL query, join customer\_orders\_lineitem with super\_mv as follows.

```
SELECT c.c_name, v.o_orderstatus
FROM customer_orders_lineitem c
JOIN super_mv v ON c.c_custkey = v.c_custkey;
```

Similarly, you can rewrite the second PartiQL query. The following example uses a filter on o\_idx = 0.

```
SELECT c.c_name, v.o_totalprice
FROM customer_orders_lineitem c
JOIN super_mv v ON c.c_custkey = v.c_custkey
WHERE v.o_idx = 0;
```

In the CREATE MATERIALIZED VIEW command, specify c\_custkey as distribution key and sort key for super\_mv. Amazon Redshift performs an efficient merge join, assuming that c\_custkey is also the distribution key and sort key of customer\_orders\_lineitem. If that isn't the case, you can specify c\_custkey as the sort key and distribution key of customer\_orders\_lineitem as follows.

```
ALTER TABLE customer_orders_lineitem
ALTER DISTKEY c_custkey, ALTER SORTKEY (c_custkey);
```

Use the EXPLAIN statement to verify that Amazon Redshift performs a merge join on the rewritten queries.

```
EXPLAIN
SELECT c.c_name, v.o_orderstatus
FROM customer_orders_lineitem c JOIN super_mv v ON c.c_custkey = v.c_custkey;

QUERY PLAN
```

```

-----+
 XN Merge Join DS_DIST_NONE (cost=0.00..34701.82 rows=1470776 width=27)
 Merge Cond: ("outer".c_custkey = "inner".c_custkey)
 -> XN Seq Scan on mv_tbl__super_mv__0 derived_table2 (cost=0.00..14999.86
 rows=1499986 width=13)
 -> XN Seq Scan on customer_orders_lineitem c (cost=0.00..999.96 rows=99996
 width=30)
 (4 rows)

```

## Creating Amazon Redshift scalar columns out of shredded data

Schemaless data stored in SUPER can affect the performance of Amazon Redshift. For instance, filter predicates or join conditions as range-restricted scans can't effectively use zone maps. Users and BI tools can use materialized views as the conventional presentation of the data and increase performance of analytical queries.

The following query scans the materialized view super\_mv and filters on o\_orderstatus.

```

SELECT c.c_name, v.o_totalprice
FROM customer_orders_lineitem c
JOIN super_mv v ON c.c_custkey = v.c_custkey
WHERE v.o_orderstatus = 'F';

```

Inspect stl\_scan to verify that Amazon Redshift can't effectively use zone maps on the range-restricted scan over o\_orderstatus.

```

SELECT slice, is_rrscan FROM stl_scan
WHERE query = pg_last_query_id() AND perm_table_name LIKE '%super_mv%';

slice | is_rrscan
-----+-----
 0 | f
 1 | f
 5 | f
 4 | f
 2 | f
 3 | f
(6 rows)

```

The following example adapts the materialized view super\_mv to create scalar columns out of the shredded data. In this case, Amazon Redshift casts o\_orderstatus from SUPER to VARCHAR. In addition, specify o\_orderstatus as the sort key for super\_mv.

```

CREATE MATERIALIZED VIEW super_mv distkey(c_custkey) sortkey(c_custkey, o_orderstatus) AS (
 SELECT c_custkey, o.o_orderstatus::VARCHAR AS o_orderstatus, o.o_totalprice, o_idx
 FROM customer_orders_lineitem c, c.c_orders o AT o_idx
);

```

After re-running the query, verify that Amazon Redshift can now use zone maps.

```

SELECT v.o_totalprice
FROM super_mv v
WHERE v.o_orderstatus = 'F';

```

You can verify that the range-restricted scan now uses zone maps as follows.

```

SELECT slice, is_rrscan FROM stl_scan

```

```
WHERE query = pg_last_query_id() AND perm_table_name LIKE '%super_mv%';

slice | is_rrscan
-----+-----
 0 | t
 1 | t
 2 | t
 3 | t
 4 | t
 5 | t
(6 rows)
```

## Limitations for using the SUPER data type with materialized views

When using SUPER data type with materialized views, observe the following limitations.

Materialized views in Amazon Redshift don't have any specific limitations with respect to PartiQL or SUPER.

For information about general SQL limitations when creating materialized views, see [Limitations \(p. 798\)](#).

For information about general SQL limitations on incremental refresh of materialized views, see [Limitations for incremental refresh \(p. 915\)](#).

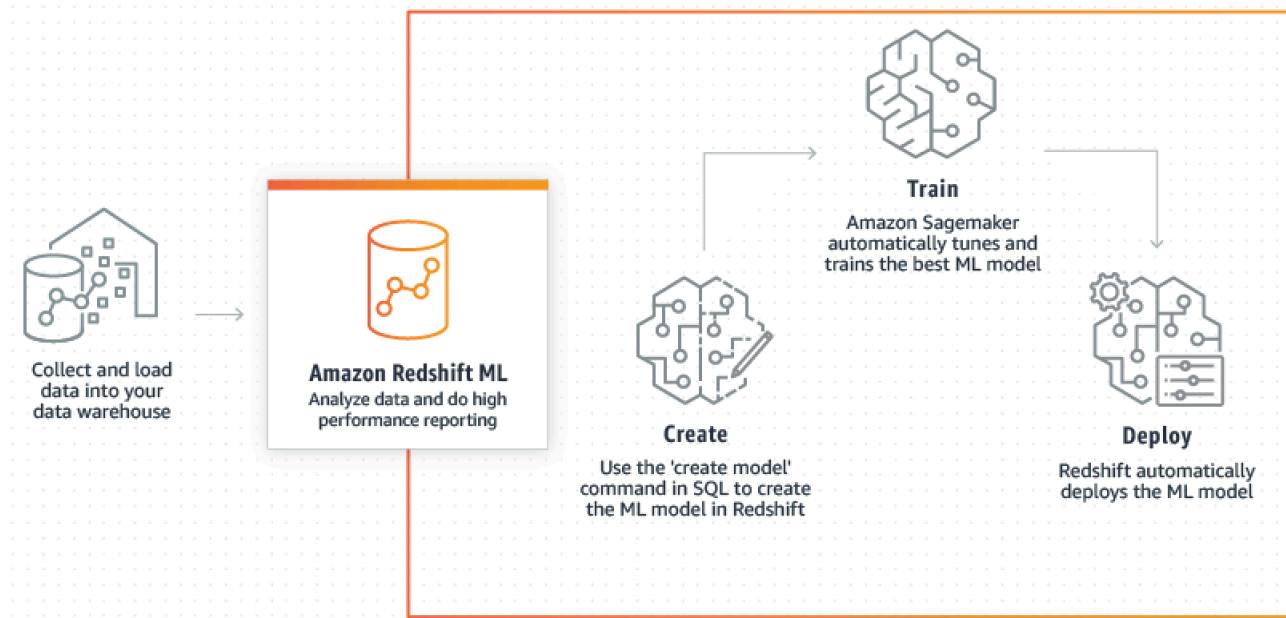
# Using machine learning in Amazon Redshift

Amazon Redshift machine learning (Amazon Redshift ML) is a robust, cloud-based service that makes it easier for analysts and data scientists of all skill levels to use machine learning technology. You provide the data that you want to train a model, and metadata associated with data inputs to Amazon Redshift. Then Amazon Redshift ML creates models that capture patterns in the input data. You can then use these models to generate predictions for new input data without incurring additional costs.

## How Amazon Redshift ML works with Amazon SageMaker

Amazon Redshift works with Amazon SageMaker Autopilot to automatically obtain the best model and make the prediction function available in Amazon Redshift.

The following diagram illustrates how Amazon Redshift ML works.



The general workflow is as follows:

1. Amazon Redshift exports the training data into Amazon S3.
2. Amazon SageMaker Autopilot preprocesses the training data. *Preprocessing* performs important functions, such as imputing missing values. It recognizes that certain columns are categorical (such as the postal code), properly formats them for training, and performs numerous other tasks. Choosing the best preprocessors to apply on the training dataset is a problem in itself, and Amazon SageMaker Autopilot automates its solution.
3. Amazon SageMaker Autopilot finds the algorithm and algorithm hyperparameters that deliver the model with the most accurate predictions.
4. Amazon Redshift registers the prediction function as a SQL function in your Amazon Redshift cluster.
5. When you run CREATE MODEL statements, Amazon Redshift uses Amazon SageMaker for training. Therefore, there is an associated cost for training your model. This is a separate line item for Amazon SageMaker in your AWS bill. You also pay for the storage used in Amazon S3 for storing your training data. Inference using models created with CREATE MODEL that you can compile and run on your

Redshift cluster aren't charged. There are no additional Amazon Redshift charges for using Amazon Redshift ML.

#### Topics

- [Machine learning overview \(p. 396\)](#)
- [Machine learning for novices and experts \(p. 398\)](#)
- [Costs for using Amazon Redshift ML \(p. 399\)](#)
- [Getting started with Amazon Redshift ML \(p. 400\)](#)

## Machine learning overview

By using Amazon Redshift ML, you can train machine learning models using SQL statements and invoke them in SQL queries for prediction.

To help you learn how to use Amazon Redshift ML, you can watch the following video: [Amazon Redshift ML](#).

For information about the prerequisites for setting up your Redshift cluster, permissions, and ownership for using Amazon Redshift ML, read the following sections. These sections also describe how simple training and predictions work in Amazon Redshift ML.

## How machine learning can solve a problem

A machine learning model generates predictions by finding patterns in your training data and then applying these patterns to new data. In machine learning, you train these models by learning the patterns that best explain your data. Then you use the models to make predictions (also called inferences) on new data. Machine learning is typically an iterative process where you can continue to improve the accuracy of the predictions by changing parameters and improving your training data. If data changes, retraining new models with the new dataset happens.

To address various business goals, there are different fundamental machine learning approaches.

## Supervised learning in Amazon Redshift ML

Amazon Redshift supports supervised learning, which is the most common approach to advanced enterprise analytics. Supervised learning is the preferred machine learning approach when you have an established set of data and an understanding of how specific input data predicts various business outcomes. These outcomes are sometimes called labels. In particular, your dataset is a table with attributes that comprise features (inputs) and targets (outputs). For example, suppose that you have a table that provides the age and postal code for past and present customers. Suppose that you also have a field "active" that is true for present customers and false for customers who have suspended their membership. The goal of supervised machine learning is to spot the patterns of age and postal code leading to customer churn, as represented by customers whose targets are "False." You can use this model to predict customers who are likely to churn, such as suspending their membership, and potentially offer retention incentives.

Amazon Redshift supports supervised learning that includes regression, binary classification, and multiclass classification. Regression refers to the problem of predicting continuous values, such as the total spending of customers. Binary classification refers to the problem of predicting one of two outcomes, such as predicting whether a customer churns or not. Multiclass classification refers to the problem of predicting one of many outcomes, such as predicting the item a customer might be interested. Data analysts and data scientists can use it to perform supervised learning to tackle problems ranging from forecasting, personalization, or customer churn prediction. You can also use supervised learning in problems such as prediction of which sales will close, revenue prediction, fraud detection, and customer life-time value prediction.

## Unsupervised learning in Amazon Redshift ML

Unsupervised learning uses machine learning algorithms to analyze and group unlabeled training data. The algorithms discover hidden patterns or groupings. The goal is to model the underlying structure or distribution in the data to learn more about the data.

Amazon Redshift supports the K-Means clustering algorithm to solve an unsupervised learning problem. This algorithm solves clustering problems where you want to discover groupings in the data. The K-Means algorithm attempts to find discrete groupings within the data. Unclassified data is grouped and partitioned based on its similarities and differences. By grouping, the K-Means algorithm iteratively determines the best centroids and assigns each member to the closest centroid. Members nearest the same centroid belong to the same group. Members of a group are as similar as possible to other members in the same group, and as different as possible from members of other groups. For example, the K-Means clustering algorithm can be used to classify cities impacted by a pandemic or classify cities based on the popularity of consumer products.

When using the K-Means algorithm, you specify an input *k* that specifies the number of clusters to find in the data. The output of this algorithm is a set of *k* centroids. Each data point belongs to one of the *k* clusters that is closest to it. Each cluster is described by its centroid. The centroid can be thought of as the multi-dimensional average of the cluster. The K-Means algorithm compares the distances to see how different the clusters are from each other. A larger distance generally indicates a greater difference between the clusters.

Preprocessing the data is important for K-Means, as it ensures that the features of the model stay on the same scale and produce reliable results. Amazon Redshift supports some K-Means preprocessors for the CREATE MODEL statement, such as StandardScaler, MinMax, and NumericPassthrough. If you don't want to apply any preprocessing for K-means, choose NumericPassthrough explicitly as a transformer. For more information about K-Means parameters, see [CREATE MODEL with K-MEANS parameters \(p. 814\)](#).

To help you learn how to perform unsupervised training with K-Means clustering, you can watch the following video: [Unsupervised training with K-Means clustering](#).

## Terms and concepts for Amazon Redshift ML

The following terms are used to describe some Amazon Redshift ML concepts:

- *Machine learning* in Amazon Redshift trains a model with one SQL command. Amazon Redshift ML and Amazon SageMaker manage all the data conversions, permissions, resource usage, and discovery of the proper model.
- *Training* is the phase when Amazon Redshift creates a machine learning model by running a specified subset of data into the model. Amazon Redshift automatically launches a training job in Amazon SageMaker and generates a model.
- *Prediction* (also called *inference*) is the use of the model in Amazon Redshift SQL queries to predict outcomes. At inference time, Amazon Redshift uses a model-based prediction function as part of a larger query to produce predictions. The predictions are computed locally, at the Redshift cluster, thus providing high throughput, low latency, and zero additional cost.
- With *bring your own model (BYOM)*, you can use a model trained outside of Amazon Redshift with Amazon SageMaker for in-database inference locally in Amazon Redshift. Amazon Redshift ML supports using BYOM in local inference.
- *Local inference* is used when models are pretrained in Amazon SageMaker, compiled by Amazon SageMaker Neo, and localized in Amazon Redshift ML. To import models that are supported for local inference to Amazon Redshift, use the CREATE MODEL command. Amazon Redshift imports the pretrained SageMaker models by calling Amazon SageMaker Neo. You compile the model there and import the compiled model into Amazon Redshift. Use local inference for faster speed and lower costs.
- *Remote inference* is used when Amazon Redshift invokes a model endpoint deployed in SageMaker. Remote inference provides the flexibility to invoke all types of custom models and deep learning models, such as TensorFlow models that you built and deployed in Amazon SageMaker.

Also important are the following:

- *Amazon SageMaker* is a fully managed machine learning service. With Amazon SageMaker, data scientists and developers can easily build, train, and directly deploy models into a production-ready hosted environment. For information about Amazon SageMaker, see [What is Amazon SageMaker](#) in the *Amazon SageMaker Developer Guide*.
- *Amazon SageMaker Autopilot* is a feature set that automatically trains and tunes the best machine learning models for classification or regression, based on your data. You maintain full control and visibility. Amazon SageMaker Autopilot supports input data in tabular format. Amazon SageMaker Autopilot provides automatic data cleaning and preprocessing, automatic algorithm selection for linear regression, binary classification, and multiclass classification. It also supports automatic hyperparameter optimization (HPO), distributed training, automatic instance, and cluster size selection. For information about Amazon SageMaker Autopilot, see [Automate model development with Amazon SageMaker Autopilot](#) in the *Amazon SageMaker Developer Guide*.

## Machine learning for novices and experts

Amazon Redshift ML enables you to train models with one single SQL CREATE MODEL command. The CREATE MODEL command creates a model that Amazon Redshift uses to generate model-based predictions with familiar SQL constructs.

Amazon Redshift ML is especially useful when you don't have expertise in machine learning, tools, languages, algorithms, and APIs. With Amazon Redshift ML, you don't have to perform the undifferentiated heavy lifting required for integrating with an external machine learning service. Amazon Redshift saves you the time to format and move data, manage permission controls, or build custom integrations, workflows, and scripts. You can easily use popular machine learning algorithms and simplify training needs that require frequent iteration from training to prediction. Amazon Redshift automatically discovers the best algorithm and tunes the best model for your problem. You can make predictions from within the Amazon Redshift cluster without the need to move data out of Amazon Redshift nor to interface with and pay for another service.

Amazon Redshift ML supports data analysts and data scientists in using machine learning. It also makes it possible for machine learning experts to use their knowledge to guide the CREATE MODEL statement to use only the aspects that they specify. By doing so, you can speed up the time that CREATE MODEL needs to find the best candidate, improve the accuracy of the model, or both.

The CREATE MODEL statement offers flexibility in how you can specify the parameters to training job. Using this flexibility, both machine learning novices or experts can choose their preferred preprocessors, algorithms, problem types, and hyperparameters. For example, a user interested in customer churn might specify for the CREATE MODEL statement that the problem type is a binary classification, which works well for customer churn. Then the CREATE MODEL statement narrows down its search for the best model into binary classification models. Even with the user choice of the problem type, there are still many options that the CREATE MODEL statement can work with. For example, the CREATE MODEL discovers and applies the best preprocessing transformations and discovers the best hyperparameter settings.

Amazon Redshift ML makes training easier by automatically finding the best model using Amazon SageMaker Autopilot. Behind the scenes, Amazon SageMaker Autopilot automatically trains and tunes the best machine learning model based on your supplied data. Amazon SageMaker Neo then compiles the training model and makes it available for prediction in your Redshift cluster. When you run a machine learning inference query using a trained model, the query can use the massively parallel processing capabilities of Amazon Redshift. At the same time, the query can use machine learning-based prediction.

- As a *machine learning beginner*, with general knowledge of different aspects of machine learning such as preprocessors, algorithms, and hyperparameters, use the CREATE MODEL statement for only the aspects that you specify. Then you can shorten the time that CREATE MODEL needs to find the

best candidate or improve the accuracy of the model. Also, you can increase the business value of the predictions by introducing additional domain knowledge such as the problem type or the objective. For example, in a customer churn scenario, if the outcome “customer is not active” is rare, then the F1 objective is often preferred to the Accuracy objective. Because high Accuracy models might predict “customer is active” all the time, this results in high accuracy but little business value. For information about F1 objectives, see [AutoMLJobObjective](#) in the *Amazon SageMaker API Reference*.

For more information about the basic options for the CREATE MODEL statement, see [Simple CREATE MODEL \(p. 805\)](#).

- As a *machine learning advanced practitioner*, you can specify the problem type and preprocessors for certain (but not all) features. Then CREATE MODEL follows your suggestions on the specified aspects. At the same time, CREATE MODEL still discovers the best preprocessors for the remaining features and the best hyperparameters. For more information about how you can constrain one or more aspects of the training pipeline, see [CREATE MODEL with user guidance \(p. 806\)](#).
- As a *machine learning expert*, you can take full control of training and hyperparameter tuning. Then the CREATE MODEL statement doesn't attempt to discover the optimal preprocessors, algorithms, and hyperparameters because you make all the choices. For more information about how to use CREATE MODEL with AUTO OFF, see [CREATE XGBoost models with AUTO OFF \(p. 809\)](#).
- As a *data engineer*, you can bring a pretrained XGBoost model in Amazon SageMaker and import it into Amazon Redshift for local inference. With bring your own model (BYOM), you can use a model trained outside of Amazon Redshift with Amazon SageMaker for in-database inference locally in Amazon Redshift. Amazon Redshift ML supports using BYOM in either local or remote inference.

For more information about how to use the CREATE MODEL statement for local or remote inference, see [Bring your own model \(BYOM\) - local inference \(p. 811\)](#).

As an Amazon Redshift ML user, you can choose any of the following options to train and deploy your model:

- Problem types, see [CREATE MODEL with user guidance \(p. 806\)](#).
- Objectives, see [CREATE MODEL with user guidance \(p. 806\)](#) or [CREATE XGBoost models with AUTO OFF \(p. 809\)](#).
- Model types, see [CREATE XGBoost models with AUTO OFF \(p. 809\)](#).
- Preprocessors, see [CREATE MODEL with user guidance \(p. 806\)](#).
- Hyperparameters, see [CREATE XGBoost models with AUTO OFF \(p. 809\)](#).
- Bring your own model (BYOM), see [Bring your own model \(BYOM\) - local inference \(p. 811\)](#).

## Costs for using Amazon Redshift ML

Amazon Redshift ML uses your existing cluster resources for prediction so you can avoid additional Amazon Redshift charges. There is no additional Amazon Redshift charge for creating or using a model. Prediction happens locally in your Redshift cluster, so you don't have to pay extra unless you need to resize your cluster. Amazon Redshift ML uses Amazon SageMaker for training your model, which does have an additional associated cost.

There is no additional charge for prediction functions that run within your Amazon Redshift cluster. The CREATE MODEL statement uses Amazon SageMaker and incurs an additional cost. The cost increases with the number of cells in your training data. The number of cells is the product of the number of records (in the training query or table times) times the number of columns. For example, when a SELECT query of the CREATE MODEL statement creates 10,000 records and 5 columns, then the number of cells it creates is 50,000.

In some cases, the training data produced by the SELECT query of the CREATE MODEL exceeds the MAX\_CELLS limit that you provided (or the default 1 million if you didn't provide a limit). In these cases,

CREATE MODEL randomly chooses approximately MAX\_CELLS (that is the “number of columns” records from the training dataset). CREATE MODEL then performs training using these randomly chosen tuples. The random sampling ensures that the reduced training dataset doesn’t have any bias. Thus, by setting the MAX\_CELLS, you can control your training costs.

When using the CREATE MODEL statement, you can use the MAX\_CELLS and MAX\_RUNTIME options to control the costs, time, and potential model accuracy.

MAX\_RUNTIME specifies the maximum amount of time the training can take in SageMaker when the AUTO ON or OFF option is used. Training jobs often complete sooner than MAX\_RUNTIME, depending on the size of the dataset. After a model is trained, Amazon Redshift does additional work in the background to compile and install your models in your cluster. Thus, CREATE MODEL can take longer than MAX\_RUNTIME to complete. However, MAX\_RUNTIME limits the amount of computation and time used in SageMaker to train your model. You can check the status of your model at any time using SHOW MODEL.

When you run CREATE MODEL with AUTO ON, Amazon Redshift ML uses SageMaker Autopilot to automatically and intelligently explore different models (or candidates) to find the best one. MAX\_RUNTIME limits the amount of time and computation spent. If MAX\_RUNTIME is set too low, there might not be enough time to explore even one candidate. If you see the error “Autopilot candidate has no models,” rerun the CREATE MODEL with a larger MAX\_RUNTIME value. For more information about this parameter, see [MaxAutoMLJobRuntimeInSeconds](#) in the *Amazon SageMaker API Reference*.

When you run CREATE MODEL with AUTO OFF, MAX\_RUNTIME corresponds to a limit on how long the training job is run in SageMaker. Training jobs often complete sooner, depending on the size of the dataset and other parameters used, such as num\_rounds in MODEL\_TYPE XGBOOST.

You can also control costs or reduce training time by specifying a smaller MAX\_CELLS value when you run CREATE MODEL. A *cell* is an entry in the database. Each row corresponds to as many cells as there are columns, which can be of fixed or varying width. MAX\_CELLS limits the number of cells, and thus the number of training examples used to train your model. By default, MAX\_CELLS is set to 1 million cells. Reducing MAX\_CELLS reduces the number of rows from the result of the SELECT query in CREATE MODEL that Amazon Redshift exports and sends to SageMaker to train a model. Reducing MAX\_CELLS thus reduces the size of the dataset used to train models both with AUTO ON and AUTO OFF. This approach helps reduce the costs and time to train models. To see information about training and billing times of a specific training job, choose **Training jobs** in Amazon SageMaker.

Increasing MAX\_RUNTIME and MAX\_CELLS often improves model quality by allowing SageMaker to explore more candidates. This way, SageMaker can take more time to train each candidate and use more data to train better models. If you want faster iteration or exploration of your dataset, use lower MAX\_RUNTIME and MAX\_CELLS. If you want improved accuracy of models, use higher MAX\_RUNTIME and MAX\_CELLS.

For more information about costs associated with various cell numbers and free trial details, see [Amazon Redshift pricing](#).

## Getting started with Amazon Redshift ML

Amazon Redshift ML makes it easy for SQL users to create, train, and deploy machine learning models using familiar SQL commands. With Amazon Redshift ML, you can use your data in your Redshift cluster to train model with Amazon SageMaker. Later, the models are localized and predictions can be made within an Amazon Redshift database. Amazon Redshift ML currently supports the machine learning algorithms XGBoost (AUTO ON and OFF) and multilayer perceptron (AUTO ON), K-Means (AUTO OFF), and Linear Learner.

### Topics

- [Cluster and configure setup for Amazon Redshift ML administration \(p. 401\)](#)

- [Using model explainability with Amazon Redshift ML \(p. 404\)](#)
  - [Amazon Redshift ML probability metrics \(p. 405\)](#)
  - [Tutorials for Amazon Redshift ML \(p. 406\)](#)

# Cluster and configure setup for Amazon Redshift ML administration

Before you work with Amazon Redshift ML, complete the cluster setup and configure permissions for using Amazon Redshift ML.

## Cluster setup for using Amazon Redshift ML

Before you work with Amazon Redshift ML, complete the following prerequisites.

As an Amazon Redshift administrator, do the following one-time setup.

To perform one-time cluster setup for Amazon Redshift ML

1. Create a Redshift cluster using the AWS Management Console or the AWS Command Line Interface (AWS CLI). Make sure to attach the AWS Identity and Access Management (IAM) policy while creating the cluster. For more information about permissions required to use Amazon Redshift ML with Amazon SageMaker, see [Permissions required to use Amazon Redshift machine learning \(ML\) with Amazon SageMaker](#).
  2. Create the IAM role required for using Amazon Redshift ML with Amazon SageMaker in one of the following ways:
    - A simple operation is to create an IAM role with AmazonS3FullAccess and AmazonSageMakerFullAccess policies for use with Amazon Redshift ML.
    - We recommend that you create an IAM role through the Amazon Redshift console that has the AmazonRedshiftAllCommandsFullAccess policy with permissions to run SQL commands, such as CREATE MODEL. Amazon Redshift uses a seamless API-based mechanism to programmatically create IAM roles in your AWS account on your behalf. Amazon Redshift automatically attaches existing AWS managed policies to the IAM role. This approach means that you can stay within the Amazon Redshift console and don't have to switch to the IAM console for role creation. For more information, see [Creating an IAM role as default for Amazon Redshift](#).

When an IAM role is created as the default for your cluster, include `redshift` as part of the resource name or use a Redshift-specific tag to tag those resources.

If your cluster has enhanced Amazon VPC routing turned on, you can use an IAM role created through the Amazon Redshift console. This IAM role has the `AmazonRedshiftAllCommandsFullAccess` policy attached and adds the following permissions to the policy. These additional permissions allow Amazon Redshift to create and delete an elastic network interface (ENI) in your account and attach it to compilation tasks running on Amazon EC2 or Amazon ECS. Doing this enables objects in your Amazon S3 buckets to be accessed only from within a virtual private cloud (VPC) with internet access blocked.

```

 "ec2:DeleteNetworkInterfacePermission",
 "ec2:DeleteNetworkInterface",
 "ec2>CreateNetworkInterfacePermission",
 "ec2>CreateNetworkInterface",
 "ec2:ModifyNetworkInterfaceAttribute"
],
 "Resource": "*"
}

```

- If you want to create an IAM role with a more restrictive policy, you can use the policy following. You can also modify this policy to meet your needs.

The Amazon S3 bucket `redshift-downloads/redshift-ml/` is the location where the sample data used for other steps and examples is stored. You can remove it if you don't need to load data from Amazon S3. Or, replace it with other Amazon S3 buckets that you use to load data into Amazon Redshift.

The `your-account-id`, `your-role`, and `your-s3-bucket` values are the ones that you specify as part of your CREATE MODEL command.

(Optional) Use the AWS KMS keys section of the sample policy if you specify an AWS KMS key while using Amazon Redshift ML. The `your-kms-key` value is the key that you use as part of your CREATE MODEL command.

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "cloudwatch:PutMetricData",
 "ecr:BatchCheckLayerAvailability",
 "ecr:BatchGetImage",
 "ecr:GetAuthorizationToken",
 "ecr:GetDownloadUrlForLayer",
 "logs>CreateLogGroup",
 "logs>CreateLogStream",
 "logs:DescribeLogStreams",
 "logs:PutLogEvents",
 "sagemaker:*Job*"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "iam:PassRole",
 "s3:AbortMultipartUpload",
 "s3:GetObject",
 "s3>DeleteObject",
 "s3:PutObject"
],
 "Resource": [
 "arn:aws:iam::<your-account-id>:role/<your-role>",
 "arn:aws:s3:::<your-s3-bucket>/*",
 "arn:aws:s3:::redshift-downloads/*"
]
 },
 {
 "Effect": "Allow",
 "Action": [
 "s3:GetBucketLocation",
 "s3>ListBucket"
]
 }
]
}

```

```

],
 "Resource": [
 "arn:aws:s3:::<your-s3-bucket>",
 "arn:aws:s3:::redshift-downloads"
]
 }
 // Optional section needed if you use AWS KMS keys.
 ,{
 "Effect": "Allow",
 "Action": [
 "kms>CreateGrant",
 "kms>Decrypt",
 "kms>DescribeKey",
 "kms>Encrypt",
 "kms>GenerateDataKey*"
],
 "Resource": [
 "arn:aws:kms:<your-region>:<your-account-id>:key/<your-kms-key>"
]
 }
}
]
}

```

3. To allow Amazon Redshift and SageMaker to assume the role to interact with other services, add the following trust policy to the IAM role.

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": [
 "redshift.amazonaws.com",
 "sagemaker.amazonaws.com"
]
 },
 "Action": "sts:AssumeRole"
 }
]
}

```

4. (Optional) Create an Amazon S3 bucket and an AWS KMS key. These are for Amazon Redshift to use to store the training data sent to Amazon SageMaker and receive the trained model from Amazon SageMaker.
5. (Optional) Create different combinations of IAM roles and Amazon S3 buckets for controlling access to different user groups.
6. (Optional) When you turn on VPC routing for your Redshift cluster, create an Amazon S3 endpoint and a SageMaker endpoint for the VPC that your Redshift cluster is in. Doing this makes it possible for traffic to run through your VPC between services during CREATE MODEL. For more information about VPC routing, see [Enhanced VPC routing in Amazon Redshift](#).

For more information about permissions required to specify a private VPC for your hyperparameter tuning job, see [Permissions required to use Amazon Redshift ML with Amazon SageMaker](#).

For information on how to use the CREATE MODEL statement to start creating models for different use cases, see [CREATE MODEL \(p. 800\)](#).

## Managing permissions and ownership

Just as with other database objects, such as tables or functions, Amazon Redshift binds creating and using ML models to access control mechanisms. There are separate permissions for creating a model that runs prediction functions.

The following examples use two user groups, `retention_analyst_grp` (model creator) and `marketing_analyst_grp` (model user) to illustrate how Amazon Redshift manages access control. The retention analyst creates machine learning models that the other set of users can use through acquired permissions.

A superuser can GRANT USER or GROUP permission to create machine learning models using the following statement.

```
GRANT CREATE MODEL TO GROUP retention_analyst_grp;
```

Users or groups with this permission can create a model in any schema in the cluster if a user has the usual CREATE permission on the SCHEMA. The machine learning model is part of the schema hierarchy in a similar way to tables, views, procedures, and user-defined functions.

Assuming a schema `demo_ml` already exists, grant the two user groups the permission on the schema as follows.

```
GRANT CREATE, USAGE ON SCHEMA demo_ml TO GROUP retention_analyst_grp;
```

```
GRANT USAGE ON SCHEMA demo_ml TO GROUP marketing_analyst_grp;
```

To let other users use your machine learning inference function, grant the EXECUTE permission. The following example uses the EXECUTE permission to grant the `marketing_analyst_grp` GROUP the permission to use the model.

```
GRANT EXECUTE ON MODEL demo_ml.customer_churn_auto_model TO GROUP marketing_analyst_grp;
```

Use the REVOKE statement with CREATE MODEL and EXECUTE to revoke those permissions from users or groups. For more information on permission control commands, see [GRANT \(p. 887\)](#) and [REVOKE \(p. 916\)](#).

## Using model explainability with Amazon Redshift ML

With model explainability in Amazon Redshift ML, you use feature importance values to help understand how each attribute in your training data contributes to the predicted result.

Model explainability helps improve your machine learning (ML) models by explaining the predictions that your models make. Model explainability helps explain how these models make predictions using a feature attribution approach.

Amazon Redshift ML incorporates model explainability to provide model explanation functionality to Amazon Redshift ML users. For more information about model explainability, see [What Is Fairness and Model Explainability for Machine Learning Predictions?](#) in the *Amazon SageMaker Developer Guide*.

Model explainability also monitors the inferences that models make in production for feature attribution drift. It also provides tools to help you generate model governance reports that you can use to inform risk and compliance teams, and external regulators.

When you specify the AUTO ON or AUTO OFF option when using the CREATE MODEL statement, after the model training job finishes, SageMaker creates the explanation output. You can use the

EXPLAIN\_MODEL function to query the explainability report in a JSON format. For more information, see [Machine learning functions \(p. 1152\)](#).

## Amazon Redshift ML probability metrics

In supervised learning problems, class labels are outcomes of predictions that use the input data. For example, if you're using a model to predict whether a customer would resubscribe to a streaming service, possible labels are likely and unlikely. Redshift ML provides the capability of probability metrics, which assign a probability to each label to indicate its likelihood. This helps you make more informed decisions based on the predicted outcomes. In Amazon Redshift ML, probability metrics are available when creating AUTO ON models with a problem type of either binary classification or multiclass classification. If you omit the AUTO ON parameter, Redshift ML assumes that the model should have AUTO ON.

### Create the model

When creating a model, Amazon Redshift automatically detects the model type and problem type. If it is a classification problem, Redshift automatically creates a second inference function that you can use to output probabilities relative to each label. This second inference function's name is your specified inference function name followed by the string \_prob. For example, if you name your inference function as `customer_churn_predict`, then the second inference function's name is `customer_churn_predict_prob`. You can then query this function to get the probabilities of each label.

```
CREATE MODEL customer_churn_model
FROM customer_activity
 PROBLEM_TYPE BINARY_CLASSIFICATION
TARGET churn
FUNCTION customer_churn_predict
IAM_ROLE {default}
AUTO ON
SETTINGS (S3_BUCKET '<DOC-EXAMPLE-BUCKET>')
```

### Get probabilities

Once the probability function is ready, running the command returns a [SUPER type](#) that contains arrays of the returned probabilities and their associated labels. For example, the result "probabilities" : [0.7, 0.3], "labels" : ["False.", "True."] means that the False label has a probability of 0.7, and the True label has a probability of 0.3.

```
SELECT customer_churn_predict_probabilities(Account_length, Area_code,
 VMail_message, Day_mins, Day_calls, Day_charge, Eve_mins, Eve_calls,
 Eve_charge, Night_mins, Night_calls, Night_charge, Intl_mins, Intl_calls,
 Intl_charge, Cust_serv_calls)
FROM customer_activity;

customer_churn_predict_probabilities

[{"probabilities" : [0.7, 0.3], "labels" : ["False.", "True."]},
 {"probabilities" : [0.8, 0.2], "labels" : ["False.", "True."]},
 {"probabilities" : [0.75, 0.25], "labels" : ["True.", "False"]}]
```

The probabilities and labels arrays are always sorted by their probabilities in descending order. You can write a query to return just the predicted label with the highest probability by unnesting the SUPER returned results of the probability function.

```
SELECT prediction.labels[0], prediction.probabilities[0]
 FROM (SELECT customer_churn_predict_probabilities(Account_length, Area_code,
 VMail_message, Day_mins, Day_calls, Day_charge, Eve_mins, Eve_calls,
```

```

 Eve_charge, Night_mins, Night_calls, Night_charge, Intl_mins, Intl_calls,
 Intl_charge, Cust_serv_calls) AS prediction
FROM customer_activity);

labels | probabilities
-----+-----
"False." | 0.7
"False." | 0.8
"True." | 0.75

```

To make the queries simpler, you can store the results of the prediction function in a table.

```

CREATE TABLE churn_auto_predict_probabilities AS
(SELECT customer_churn_predict_probabilities(Account_length, Area_code,
VMail_message, Day_mins, Day_calls, Day_charge, Eve_mins, Eve_calls,
Eve_charge, Night_mins, Night_calls, Night_charge, Intl_mins,
Intl_calls, Intl_charge, Cust_serv_calls) AS prediction
FROM customer_activity);

```

You can query the table with the results to return only predictions that have a probability higher than 0.7.

```

SELECT prediction.labels[0], prediction.probabilities[0]
FROM churn_auto_predict_probabilities
WHERE prediction.probabilities[0] > 0.7;

labels | probabilities
-----+-----
"False." | 0.8
"True." | 0.75

```

Using index notation, you can get the probability of a specific label. The following example returns probabilities of all the `True.` labels.

```

SELECT label, index, p.prediction.probabilities[index]
FROM churn_auto_predict_probabilities p, p.prediction.labels AS label AT index
WHERE label='True.';

label | index | probabilities
-----+-----+-----
"True." | 0 | 0.3
"True." | 0 | 0.2
"True." | 0 | 0.75

```

The following example returns all rows that have a `True.` label with a probability greater than 0.7, indicating that the customer is likely to churn.

```

SELECT prediction.labels[0], prediction.probabilities[0]
FROM churn_auto_predict_probabilities
WHERE prediction.probabilities[0] > 0.7 AND prediction.labels[0] = "True./";

labels | probabilities
-----+-----
"True." | 0.75

```

## Tutorials for Amazon Redshift ML

You can use Amazon Redshift ML to train machine learning models using SQL statements, and then invoke the models in SQL queries for prediction. Machine learning in Amazon Redshift trains a model

with one SQL command. Amazon Redshift automatically launches a training job in Amazon SageMaker and generates a model. Once a model is created, you can perform predictions in Amazon Redshift using the model's prediction function.

Follow the steps in these tutorials to learn about Amazon Redshift ML features:

- [Tutorial: Building customer churn models \(p. 407\)](#)
- [Tutorial: Building remote inference models \(p. 413\)](#)
- [Tutorial: Building K-means clustering models \(p. 416\)](#)
- [Tutorial: Building multi-class classification models \(p. 423\)](#)
- [Tutorial: Building XGBoost models \(p. 429\)](#)
- [Tutorial: Building regression models \(p. 434\)](#)
- [Tutorial: Building regression models with linear learner \(p. 442\)](#)
- [Tutorial: Building multi-class classification models with linear learner \(p. 448\)](#)

## Tutorial: Building customer churn models

In this tutorial, you use Amazon Redshift ML to create a customer churn model with the CREATE MODEL command, and run prediction queries for user scenarios. Then, you implement queries using the SQL function that the CREATE MODEL command generates.

You can use a simple CREATE MODEL command to export training data, train a model, import the model, and prepare an Amazon Redshift prediction function. Use the CREATE MODEL statement to specify training data either as a table or SELECT statement.

This example uses historical information to construct a machine learning model of a mobile operator's customer churn. First, SageMaker trains your machine learning model and then tests your model using the profile information of an arbitrary customer. After the model is validated, Amazon SageMaker deploys the model and the prediction function to Amazon Redshift. You can use the prediction function to predict whether a customer is going to churn or not.

### Use case examples

You can solve other binary classification problems using Amazon Redshift ML, such as predicting if a sales lead will close or not. You could also predict whether a financial transaction is fraudulent or not.

#### Tasks

- Prerequisites
- Step 1: Load the data from Amazon S3 to Amazon Redshift
- Step 2: Create the machine learning model
- Step 3: Perform predictions with the model

### Prerequisites

To complete this tutorial, you must have the following prerequisites:

- You must set up an Amazon Redshift cluster for Amazon Redshift ML. To do so, use the documentation for [Cluster and configure setup for Amazon Redshift ML administration](#).
- The Amazon Redshift cluster that you use to create the model, and the Amazon S3 bucket that you use to stage the training data and store the model artifacts must be in the same AWS Region.
- To download the SQL commands and the sample dataset used in this documentation, do one of the following:
  - Download the [SQL commands](#), [Customer activity file](#), and [Abalone file](#).

- Using the AWS CLI for Amazon S3, run the following command. You can use your own target path.

```
aws s3 cp s3://redshift-downloads/redshift-ml/tutorial-scripts/redshift-ml-tutorial.sql </target/path>
aws s3 cp s3://redshift-downloads/redshift-ml/customer_activity/customer_activity.csv </target/path>
aws s3 cp s3://redshift-downloads/redshift-ml/abalone_xgb/abalone_xgb.csv </target/path>
```

## Step 1: Load the data from Amazon S3 to Amazon Redshift

Use the [Amazon Redshift query editor v2](#) to edit and run queries and visualize results.

Running the following queries creates a table named `customer_activity` and ingests the sample dataset from Amazon S3.

```
DROP TABLE IF EXISTS customer_activity;

CREATE TABLE customer_activity (
state varchar(2),
account_length int,
area_code int,
phone varchar(8),
intl_plan varchar(3),
vMail_plan varchar(3),
vMail_message int,
day_mins float,
day_calls int,
day_charge float,
total_charge float,
eve_mins float,
eve_calls int,
eve_charge float,
night_mins float,
night_calls int,
night_charge float,
intl_mins float,
intl_calls int,
intl_charge float,
cust_serv_calls int,
churn varchar(6),
record_date date
);

COPY customer_activity
FROM 's3://redshift-downloads/redshift-ml/customer_activity/'
REGION 'us-east-1' IAM_ROLE default
FORMAT AS CSV IGNOREHEADER 1;
```

## Step 2: Create the machine learning model

Churn is our target input in this model. All other inputs for the model are attributes that help to create a function to predict churn.

The following example uses the CREATE MODEL operation to deliver a model that predicts whether a customer will be active, using inputs such as the customer's age, postal code, spending, and cases. In the following example, replace **DOC-EXAMPLE-BUCKET** with your own Amazon S3 bucket.

```
CREATE MODEL customer_churn_auto_model
FROM
```

```

(
 SELECT state,
 account_length,
 area_code,
 total_charge/account_length AS average_daily_spend,
 cust_serv_calls/account_length AS average_daily_cases,
 churn
 FROM customer_activity
 WHERE record_date < '2020-01-01'
)
TARGET churn FUNCTION ml_fn_customer_churn_auto
IAM_ROLE default SETTINGS (
 S3_BUCKET '<DOC-EXAMPLE-BUCKET>'
);

```

The SELECT query in the preceding example creates the training data. The TARGET clause specifies which column is the machine learning label that the CREATE MODEL operation uses to learn how to predict. The target column “churn” indicates whether the customer still has an active membership or has suspended the membership. The S3\_BUCKET field is the name of the Amazon S3 bucket that you previously created. The Amazon S3 bucket is used to share training data and artifacts between Amazon Redshift and Amazon SageMaker. The remaining columns are the features that are used for the prediction.

For a summary of the syntax and features of a basic use case of the CREATE MODEL command, see [Simple CREATE MODEL](#).

#### Add permissions for server-side encryption (optional)

Amazon Redshift by default uses Amazon SageMaker Autopilot for training. In particular, Amazon Redshift securely exports the training data to the customer-specified Amazon S3 bucket. If you don't specify a KMS\_KEY\_ID, then the data is encrypted using server-side encryption SSE-S3 by default.

When you encrypt your input using server-side encryption with a AWS KMS managed key (SSE-MMS), then add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "kms:Encrypt",
 "kms:Decrypt"
]
}
```

For more information about Amazon SageMaker roles, see [Amazon SageMaker roles](#) in the *Amazon SageMaker Developer Guide*.

#### Check the status of model training (optional)

You can use the SHOW MODEL command to know when your model is ready.

Use the following operation to check the status of the model.

```
SHOW MODEL customer_churn_auto_model;
```

The following is an example of the output of the previous operation.

```
+-----+
+-----+
+
```

| Key                                                                        | Value                                                   |
|----------------------------------------------------------------------------|---------------------------------------------------------|
|                                                                            |                                                         |
| +-----                                                                     |                                                         |
| +-----                                                                     |                                                         |
| Model Name                                                                 |                                                         |
| customer_churn_auto_model                                                  |                                                         |
| Schema Name                                                                |                                                         |
| public                                                                     |                                                         |
| Owner                                                                      |                                                         |
| awsuser                                                                    |                                                         |
| Creation Time                                                              |                                                         |
| Tue, 14.06.2022 17:15:52                                                   |                                                         |
| Model State                                                                |                                                         |
| TRAINING                                                                   |                                                         |
|                                                                            |                                                         |
| TRAINING DATA:                                                             |                                                         |
|                                                                            |                                                         |
| Query                                                                      | SELECT STATE, ACCOUNT_LENGTH, AREA_CODE, TOTAL_CHARGE / |
| ACCOUNT_LENGTH AS AVERAGE_DAILY_SPEND, CUST_SERV_CALLS / ACCOUNT_LENGTH AS |                                                         |
| AVERAGE_DAILY_CASES, CHURN                                                 |                                                         |
| FROM CUSTOMER_ACTIVITY                                                     |                                                         |
|                                                                            |                                                         |
| WHERE RECORD_DATE < '2020-01-01'                                           |                                                         |
| Target Column                                                              |                                                         |
| CHURN                                                                      |                                                         |
|                                                                            |                                                         |
| PARAMETERS:                                                                |                                                         |
|                                                                            |                                                         |
| Model Type                                                                 |                                                         |
| auto                                                                       |                                                         |
| Problem Type                                                               |                                                         |
|                                                                            |                                                         |
| Objective                                                                  |                                                         |
|                                                                            |                                                         |
| AutoML Job Name                                                            |                                                         |
| redshiftml-20220614171552640901                                            |                                                         |
|                                                                            |                                                         |
| Function Name                                                              |                                                         |
| ml_fn_customer_churn_auto                                                  |                                                         |
|                                                                            |                                                         |
| Function Parameters                                                        |                                                         |
| account_length area_code average_daily_spend average_daily_cases           | state                                                   |
|                                                                            |                                                         |
| Function Parameter Types                                                   |                                                         |
| varchar int4 int4 float8 int4                                              |                                                         |
|                                                                            |                                                         |

```

| IAM Role |
| default-aws-iam-role |
| S3 Bucket |
| DOC-EXAMPLE-BUCKET |
| Max Runtime |
| 5400 |
+-----+
+-----+
+

```

When the model training is complete, the `model_state` variable becomes `Model is Ready`, and the prediction function becomes available.

### Step 3: Perform predictions with the model

You can use SQL statements to view the predictions made by the prediction model. In this example, the prediction function created by the CREATE MODEL operation is named `ml_fn_customer_churn_auto`. The input arguments for the prediction function correspond to the types of the features, such as `varchar` for the `state` and `integer` for `account_length`. The output of the prediction function is the same type as the `TARGET` column of the CREATE MODEL statement.

1. You trained the model on data from before 2020-01-01, so now you use the prediction function on the testing set. The following query displays the predictions of whether customers who signed up after 2020-01-01 will go through churn or not.

```

SELECT
 phone,
 ml_fn_customer_churn_auto(
 state,
 account_length,
 area_code,
 total_charge / account_length,
 cust_serv_calls / account_length
) AS active
FROM
 customer_activity
WHERE
 record_date > '2020-01-01';

```

2. The following example uses the same prediction function for a different use case. In this case, Amazon Redshift predicts the proportion of churners and non-churners among customers from different states where the record date is greater than 2020-01-01.

```

WITH predicted AS (
 SELECT
 state,
 ml_fn_customer_churn_auto(
 state,
 account_length,
 area_code,
 total_charge / account_length,
 cust_serv_calls / account_length
) :: varchar(6) AS active
 FROM
 customer_activity
 WHERE
 record_date > '2020-01-01'
)
SELECT

```

```

state,
SUM(
CASE
 WHEN active = 'True.' THEN 1
 ELSE 0
END
) AS churners,
SUM(
CASE
 WHEN active = 'False.' THEN 1
 ELSE 0
END
) AS nonchurners,
COUNT(*) AS total_per_state
FROM
predicted
GROUP BY
state
ORDER BY
state;

```

3. The following example uses the prediction function for the use case of predicting the percentage of customers who churn in a state. In this case, Amazon Redshift predicts the churn percentage where the record date is greater than 2020-01-01.

```

WITH predicted AS (
SELECT
state,
ml_fn_customer_churn_auto(
state,
account_length,
area_code,
total_charge / account_length,
cust_serv_calls / account_length
):: varchar(6) AS active
FROM
customer_activity
WHERE
record_date > '2020-01-01'
)
SELECT
state,
CAST((CAST((SUM(
CASE
 WHEN active = 'True.' THEN 1
 ELSE 0
END
)) AS FLOAT) / CAST(COUNT(*) AS FLOAT)) AS DECIMAL (3, 2)) AS pct_churn,
COUNT(*) AS total_customers_per_state
FROM
predicted
GROUP BY
state
ORDER BY
3 DESC;

```

## Related topics

For more information about Amazon Redshift ML, see the following documentation:

- [Costs for using Amazon RedshiftML](#)
- [CREATE MODEL command](#)

- [EXPLAIN\\_MODEL function](#)

For more information about machine learning, see the following documentation:

- [Machine learning overview](#)
- [Machine learning for novices and experts](#)
- [What Is Fairness and Model Explainability for Machine Learning Predictions?](#)

## Tutorial: Building remote inference models

The following tutorial goes over the steps of how to create a [Random Cut Forest model](#) that has been previously trained and deployed in Amazon SageMaker, outside of Amazon Redshift. The Random Cut Forest algorithm detects anomalous data points within a dataset. Creating a model with remote inference allows you to bring your Random Cut Forest SageMaker model into Amazon Redshift. Then, in Amazon Redshift, you use SQL to perform predictions on a remote SageMaker endpoint.

You can use a CREATE MODEL command to import a machine learning model from an Amazon SageMaker endpoint and prepare an Amazon Redshift prediction function. When using the CREATE MODEL operation, you provide the SageMaker machine learning model's endpoint name.

In this tutorial, you create an Amazon Redshift machine learning model using a SageMaker model endpoint. Once your machine learning model is ready, you can use it to perform predictions in Amazon Redshift. First, you train and create an endpoint in Amazon SageMaker, and then you get the endpoint name. Then, you use the CREATE MODEL command to create a model with Amazon Redshift ML. Finally, you perform predictions on the model using the prediction function that the CREATE MODEL command generates.

### Use case examples

You can use Random Cut Forest models and remote inference for anomaly detection in other datasets, such as predicting a rapid increase or decrease in e-commerce transactions. You could also predict significant changes in weather or seismic activity.

### Tasks

- Prerequisites
- Step 1: Deploy the Amazon SageMaker model
- Step 2: Get the SageMaker model endpoint
- Step 3: Load the data from Amazon S3 to Amazon Redshift
- Step 4: Create a model with Amazon Redshift ML
- Step 5: Perform predictions with the model

### Prerequisites

To complete this tutorial, you must have the following prerequisites:

- You have completed the [Administrative setup](#) for Amazon Redshift ML.
- You have downloaded the [NYC taxi dataset](#), [created an Amazon S3 bucket](#), and [uploaded the data into the Amazon S3 bucket](#).
- You must train, deploy the SageMaker model and endpoint, and get the name of the SageMaker endpoint. Use [this AWS CloudFormation template](#) to provision all the SageMaker resources in your AWS account automatically.

## Step 1: Deploy the Amazon SageMaker model

1. To deploy the model, go to the Amazon SageMaker console, choose **Notebook instances** under **Notebook** in the navigation pane.
2. Choose **Open Jupyter** for the Jupyter notebook that was created by the CloudFormation template.
3. Choose `bring-your-own-model-remote-inference.ipynb`.
4. Set up the parameters to store the training input and output in Amazon S3 by replacing the following lines with your Amazon S3 bucket and prefix.

```
data_location=f"s3://{{bucket}}/{{prefix}}/",
output_path=f"s3://{{bucket}}/{{prefix}}/output",
```

5. Choose the **fast-forward** button to run all cells.

## Step 2: Get the SageMaker model endpoint

On the Amazon SageMaker console, under **Inference** in the navigation pane, choose **Endpoints** and find your model name. You must copy your model's endpoint name when you create the remote inference model in Amazon Redshift.

## Step 3: Load the data from Amazon S3 to Amazon Redshift

Use the [Amazon Redshift query editor v2](#) to run the following SQL commands in Amazon Redshift. These commands drop the `rcf_taxi_data` table if it exists, create a table of the same name, and load the sample dataset into the table.

```
DROP TABLE IF EXISTS public.rcf_taxi_data CASCADE;

CREATE TABLE public.rcf_taxi_data (ride_timestamp timestamp, nbr_passengers int);

COPY public.rcf_taxi_data
FROM
 's3://sagemaker-sample-files/datasets/tabular/anomaly_benchmark_taxi/NAB_nyc_taxi.csv'
IAM_ROLE default
IGNOREHEADER 1
FORMAT AS CSV;
```

## Step 4: Create a model with Amazon Redshift ML

Run the following query to create a model in Amazon Redshift ML using the SageMaker model endpoint you got in the previous step. Replace `<randomcutforest-xxxxxxxxx>` with your own SageMaker endpoint's name.

```
CREATE MODEL public.remote_random_cut_forest
FUNCTION remote_fn_rcf(int)
RETURNS decimal(10, 6) SAGEMAKER '<randomcutforest-xxxxxxxxx>' IAM_ROLE default;
```

### Check the model status (optional)

You can use the SHOW MODEL command to know when your model is ready.

To check the model status, use the following SHOW MODEL operation.

```
SHOW MODEL public.remote_random_cut_forest
```

The output shows the SageMaker endpoint and function name.

|                          |                            |
|--------------------------|----------------------------|
| Model Name               | remote_random_cut_forest   |
| Schema Name              | public                     |
| Owner                    | awsuser                    |
| Creation Time            | Wed, 15.06.2022 17:58:21   |
| Model State              | READY                      |
| PARAMETERS:              |                            |
| Endpoint                 | <randomcutforest-xxxxxxxx> |
| Function Name            | remote_fn_rcf              |
| Inference Type           | Remote                     |
| Function Parameter Types | int4                       |
| IAM Role                 | default-aws-iam-role       |

## Step 5: Perform predictions with the model

The Amazon SageMaker Random Cut Forest algorithm is designed to detect anomalous data points within a dataset. In this example, your model is designed to detect spikes in taxi rides due to important events. You can use the model to predict anomalous events by generating an anomaly score for each data point.

Use the following query to compute anomaly scores across the entire taxi dataset. Note that you reference the function that you used in your CREATE MODEL statement in the previous step.

```
SELECT
 ride_timestamp,
 nbr_passengers,
 public.remote_fn_rcf(nbr_passengers) AS score
FROM
 public.rcf_taxi_data;
```

### Check for high and low anomalies (optional)

Run the following query to find any data points with scores greater than three standard deviations from the mean score.

```
WITH score_cutoff AS (
 SELECT
 STDDEV(public.remote_fn_rcf(nbr_passengers)) AS std,
 AVG(public.remote_fn_rcf(nbr_passengers)) AS mean,
 (mean + 3 * std) AS score_cutoff_value
 FROM
 public.rcf_taxi_data
)
SELECT
 ride_timestamp,
 nbr_passengers,
 public.remote_fn_rcf(nbr_passengers) AS score
FROM
 public.rcf_taxi_data
WHERE
 score > (
 SELECT
 score_cutoff_value
 FROM
 score_cutoff
)
ORDER BY
 2 DESC;
```

Run the following query to find any data points with scores greater than three standard deviations from the mean score.

```
WITH score_cutoff AS (
 SELECT
 STDDEV(public.remote_fn_rcf(nbr_passengers)) AS std,
 AVG(public.remote_fn_rcf(nbr_passengers)) AS mean,
 (mean - 3 * std) AS score_cutoff_value
 FROM
 public.rcf_taxi_data
)
SELECT
 ride_timestamp,
 nbr_passengers,
 public.remote_fn_rcf(nbr_passengers) AS score
FROM
 public.rcf_taxi_data
WHERE
 score < (
 SELECT
 score_cutoff_value
 FROM
 score_cutoff
)
ORDER BY
 2 DESC;
```

## Related topics

For more information about Amazon Redshift ML, see the following documentation:

- [Costs for using Amazon Redshift ML](#)
- [CREATE MODEL operation](#)
- [EXPLAIN\\_MODEL function](#)

For more information about machine learning, see the following documentation:

- [Machine learning overview](#)
- [Machine learning for novices and experts](#)
- [What Is Fairness and Model Explainability for Machine Learning Predictions?](#)

## Tutorial: Building K-means clustering models

In this tutorial, you use Amazon Redshift ML to create, train, and deploy a machine learning model based on the [K-means algorithm](#). This algorithm solves clustering problems where you want to discover groupings in the data. K-means helps in grouping data that has not been labeled yet. To learn more about K-means clustering, see [How K-means Clustering Works](#) in the Amazon SageMaker Developer Guide.

You will use a CREATE MODEL operation to create a K-means model from a Amazon Redshift cluster. You can use a CREATE MODEL command to export training data, train a model, import the model, and prepare an Amazon Redshift prediction function. Use the CREATE MODEL operation to specify training data either as a table or a SELECT statement.

In this tutorial, you use K-means on the [Global Database of Events, Language, and Tone \(GDELT\)](#) dataset, which monitors world news across the world, and the data is stored for every second of every day. K-means will group events that have similar tone, actors, or locations. The data is stored as multiple files on Amazon Simple Storage Service, in two different folders. The folders are historical, which cover the

years 1979–2013, and daily updates, which cover the years 2013 and later. For this example, we use the historical format and bring in 1979 data.

## Use case examples

You can solve other clustering problems with Amazon Redshift ML, such as grouping customers who have similar viewing habits on a streaming service. You could also use Redshift ML to predict the optimal number of shipping centers for a delivery service.

### Tasks

- Prerequisites
- Step 1: Load the data from Amazon S3 to Amazon Redshift
- Step 2: Create the machine learning model
- Step 3: Perform predictions with the model

### Prerequisites

To complete this tutorial, you must complete the [Administrative setup](#) for Amazon Redshift ML.

## Step 1: Load the data from Amazon S3 to Amazon Redshift

1. Use the [Amazon Redshift query editor v2](#) to run the following query. The query drops the gdelt\_data table in the public schema if it exists and creates a table of the same name in the public schema.

```
DROP TABLE IF EXISTS gdelt_data CASCADE;

CREATE TABLE gdelt_data (
 GlobalEventId bigint,
 SqlDate bigint,
 MonthYear bigint,
 Year bigint,
 FractionDate double precision,
 Actor1Code varchar(256),
 Actor1Name varchar(256),
 Actor1CountryCode varchar(256),
 Actor1KnownGroupCode varchar(256),
 Actor1EthnicCode varchar(256),
 Actor1Religion1Code varchar(256),
 Actor1Religion2Code varchar(256),
 Actor1Type1Code varchar(256),
 Actor1Type2Code varchar(256),
 Actor1Type3Code varchar(256),
 Actor2Code varchar(256),
 Actor2Name varchar(256),
 Actor2CountryCode varchar(256),
 Actor2KnownGroupCode varchar(256),
 Actor2EthnicCode varchar(256),
 Actor2Religion1Code varchar(256),
 Actor2Religion2Code varchar(256),
 Actor2Type1Code varchar(256),
 Actor2Type2Code varchar(256),
 Actor2Type3Code varchar(256),
 IsRootEvent bigint,
 EventCode bigint,
 EventBaseCode bigint,
 EventRootCode bigint,
 QuadClass bigint,
 GoldsteinScale double precision,
```

```

NumMentions bigint,
NumSources bigint,
NumArticles bigint,
AvgTone double precision,
Actor1Geo_Type bigint,
Actor1Geo_FullName varchar(256),
Actor1Geo_CountryCode varchar(256),
Actor1Geo ADM1Code varchar(256),
Actor1Geo_Lat double precision,
Actor1Geo_Long double precision,
Actor1Geo_FeatureID bigint,
Actor2Geo_Type bigint,
Actor2Geo_FullName varchar(256),
Actor2Geo_CountryCode varchar(256),
Actor2Geo ADM1Code varchar(256),
Actor2Geo_Lat double precision,
Actor2Geo_Long double precision,
Actor2Geo_FeatureID bigint,
ActionGeo_Type bigint,
ActionGeo_FullName varchar(256),
ActionGeo_CountryCode varchar(256),
ActionGeo ADM1Code varchar(256),
ActionGeo_Lat double precision,
ActionGeo_Long double precision,
ActionGeo_FeatureID bigint,
DATEADDED bigint
);

```

2. The following query loads the sample data into the gdelt\_data table.

```

COPY gdelt_data
FROM 's3://gdelt-open-data/events/1979.csv'
REGION 'us-east-1'
IAM_ROLE default
CSV
DELIMITER '\t';

```

### Examine the training data (optional)

To see what data your model will be trained on, use the following query.

```

SELECT
 AvgTone,
 EventCode,
 NumArticles,
 Actor1Geo_Lat,
 Actor1Geo_Long,
 Actor2Geo_Lat,
 Actor2Geo_Long
FROM
 gdelt_data LIMIT 100;

```

## Step 2: Create the machine learning model

The following example uses the CREATE MODEL command to create a model that groups the data into seven clusters. The K value is the number of clusters that your data points are divided into. The model classifies your data points into clusters where data points are more similar to each other. By clustering the data points into groups, the K-Means algorithm iteratively determines the best cluster center. The algorithm then assigns each data point to the closest cluster center. Members nearest the same cluster center belong to the same group. Members of a group are as similar as possible to other members in the same group, and as different as possible from members of other groups. The K value is subjective and

depends on methods that measure the similarities among data points. You can change the K value to smooth out cluster sizes if the clusters are unevenly distributed.

In the following example, replace *DOC-EXAMPLE-BUCKET* with your own Amazon S3 bucket.

```
CREATE MODEL news_data_clusters
FROM
(
 SELECT
 AvgTone,
 EventCode,
 NumArticles,
 Actor1Geo_Lat,
 Actor1Geo_Long,
 Actor2Geo_Lat,
 Actor2Geo_Long
 FROM
 gdelt_data
) FUNCTION news_monitoring_cluster
IAM_ROLE default
AUTO OFF
MODEL_TYPE KMEANS
PREPROCESSORS 'none'
HYPERPARAMETERS DEFAULT
EXCEPT
(K '7')
SETTINGS (S3_BUCKET '<DOC-EXAMPLE-BUCKET>');
```

#### Check the status of model training (optional)

You can use the SHOW MODEL command to know when your model is ready.

To check the model status, use the following SHOW MODEL operation and find if the Model State is Ready.

```
SHOW MODEL NEWS_DATA_CLUSTERS;
```

When the model is ready, the output of the previous operation should show that the Model State is Ready. The following is an example of the output of the SHOW MODEL operation.

|                                            |  |  |
|--------------------------------------------|--|--|
| +-----                                     |  |  |
| +-----                                     |  |  |
| +-----                                     |  |  |
| Model Name     news_data_clusters          |  |  |
| +-----                                     |  |  |
| +-----                                     |  |  |
| +-----                                     |  |  |
| Schema Name     public                     |  |  |
| Owner     awsuser                          |  |  |
| Creation Time     Fri, 17.06.2022 16:32:19 |  |  |
| Model State     READY                      |  |  |
| train:msd     2973.822754                  |  |  |
| train:progress     100.000000              |  |  |
| train:throughput     237114.875000         |  |  |

|        |                                        |  |                                                        |                                |
|--------|----------------------------------------|--|--------------------------------------------------------|--------------------------------|
|        | Estimated Cost                         |  |                                                        | 0.004983                       |
|        |                                        |  |                                                        |                                |
|        | TRAINING DATA:                         |  |                                                        |                                |
|        | Query                                  |  | SELECT AVGTONE, EVENTCODE, NUMARTICLES, ACTOR1GEO_LAT, |                                |
|        |                                        |  | ACTOR1GEO_LONG, ACTOR2GEO_LAT, ACTOR2GEO_LONG          | FROM GDELT_DATA                |
|        |                                        |  |                                                        |                                |
|        | PARAMETERS:                            |  |                                                        |                                |
|        | Model Type                             |  |                                                        | kmeans                         |
|        | Training Job Name                      |  |                                                        |                                |
|        | redshiftml-20220617163219978978-kmeans |  |                                                        |                                |
|        | Function Name                          |  |                                                        | news_monitoring_cluster        |
|        | Function Parameters                    |  | avgtone eventcode numarticles actor1geo_lat            |                                |
|        |                                        |  | actor1geo_long actor2geo_lat actor2geo_long            |                                |
|        | Function Parameter Types               |  |                                                        | float8 int8 int8 float8 float8 |
|        |                                        |  | float8 float8                                          |                                |
|        | IAM Role                               |  |                                                        | default-aws-iam-role           |
|        | S3 Bucket                              |  |                                                        | DOC-EXAMPLE-BUCKET             |
|        | Max Runtime                            |  |                                                        | 5400                           |
|        |                                        |  |                                                        |                                |
|        | HYPERPARAMETERS:                       |  |                                                        |                                |
|        | feature_dim                            |  |                                                        | 7                              |
|        | k                                      |  |                                                        | 7                              |
| +----- |                                        |  |                                                        |                                |
| +----- |                                        |  |                                                        |                                |
| +      |                                        |  |                                                        |                                |

## Step 3: Perform predictions with the model

### Identify the clusters

You can find discrete groupings identified in the data by your model, otherwise known as clusters. A cluster is the set of data points that is closer to its cluster center than any other cluster center. Since the K value represents the number of clusters in the model, it also represents the number of cluster centers. The following query identifies the clusters by showing the cluster associated with each globaleventid.

```
SELECT
 globaleventid,
 news_monitoring_cluster (
 AvgTone,
 EventCode,
 NumArticles,
 Actor1Geo_Lat,
 Actor1Geo_Long,
 Actor2Geo_Lat,
 Actor2Geo_Long
) AS cluster
FROM
```

```
gdelt_data;
```

### Check the distribution of data

You can check the distribution of data across clusters to see if the K value that you chose caused the data to be somewhat evenly distributed. Use the following query to determine if the data is evenly distributed across your clusters.

```
SELECT
 events_cluster,
 COUNT(*) AS nbr_events
FROM
(
 SELECT
 globaleventid,
 news_monitoring_cluster(
 AvgTone,
 EventCode,
 NumArticles,
 Actor1Geo_Lat,
 Actor1Geo_Long,
 Actor2Geo_Lat,
 Actor2Geo_Long
) AS events_cluster
 FROM
 gdelt_data
)
GROUP BY
 1;
```

Note that you can change the K value to smooth out cluster sizes if the clusters are unevenly distributed.

### Determine the cluster centers

A data point is closer to its cluster center than it is to any other cluster center. Thus, finding the cluster centers helps you define the clusters.

Run the following query to determine the centers of the clusters based on the number of articles by event code.

```
SELECT
 news_monitoring_cluster (
 AvgTone,
 EventCode,
 NumArticles,
 Actor1Geo_Lat,
 Actor1Geo_Long,
 Actor2Geo_Lat,
 Actor2Geo_Long
) AS events_cluster,
 eventcode,
 SUM(numArticles) AS numArticles
FROM
 gdelt_data
GROUP BY
 1,
 2;
```

### Show information about data points in a cluster

Use the following query to return the data for the points assigned to the fifth cluster. The selected articles must have two actors.

```

SELECT
 news_monitoring_cluster (
 AvgTone,
 EventCode,
 NumArticles,
 Actor1Geo_Lat,
 Actor1Geo_Long,
 Actor2Geo_Lat,
 Actor2Geo_Long
) AS events_cluster,
 eventcode,
 actor1name,
 actor2name,
 SUM(numarticles) AS totalarticles
FROM
 gdelt_data
WHERE
 events_cluster = 5
 AND actor1name <> ''
 AND actor2name <> ''
GROUP BY
 1,
 2,
 3,
 4
ORDER BY
 5 desc;

```

### Show data about events with actors of the same ethnic code

The following query counts the number of articles written about events with a positive tone. The query also requires that the two actors have the same ethnic code and it returns which cluster each event is assigned to.

```

SELECT
 news_monitoring_cluster (
 AvgTone,
 EventCode,
 NumArticles,
 Actor1Geo_Lat,
 Actor1Geo_Long,
 Actor2Geo_Lat,
 Actor2Geo_Long
) AS events_cluster,
 SUM(numarticles) AS total_articles,
 eventcode AS event_code,
 Actor1EthnicCode AS ethnic_code
FROM
 gdelt_data
WHERE
 Actor1EthnicCode = Actor2EthnicCode
 AND Actor1EthnicCode <> ''
 AND Actor2EthnicCode <> ''
 AND AvgTone > 0
GROUP BY
 1,
 3,
 4
HAVING
 (total_articles) > 4
ORDER BY
 1,
 2 ASC;

```

## Related topics

For more information about Amazon Redshift ML, see the following documentation:

- [Costs for using Amazon Redshift ML](#)
- [CREATE MODEL operation](#)
- [EXPLAIN\\_MODEL function](#)

For more information about machine learning, see the following documentation:

- [Machine learning overview](#)
- [Machine learning for novices and experts](#)
- [What Is Fairness and Model Explainability for Machine Learning Predictions?](#)

## Tutorial: Building multi-class classification models

In this tutorial, you use Amazon Redshift ML to create a machine learning model that solves multi-class classification problems. The multi-class classification algorithm classifies data points into one of three or more classes. Then, you implement queries using the SQL function that the CREATE MODEL command generates.

You can use a CREATE MODEL command to export training data, train a model, import the model, and prepare an Amazon Redshift prediction function. Use the CREATE MODEL operation to specify training data either as a table or a SELECT statement.

To follow along with the tutorial, you use the public dataset [E-Commerce Sales Forecast](#), which includes sales data of an online UK retailer. The model you generate will target the most active customers for a special customer loyalty program. With multi-class classification, you can use the model to predict how many months a customer will be active over a 13-month period. The prediction function designates customers who are predicted to be active for 7 or more months for admission to the program.

### Use case examples

You can solve other multi-class classification problems with Amazon Redshift ML, such as predicting the best-selling product from a product line. You could also predict which fruit an image contains, such as selecting apples or pears or oranges.

### Tasks

- Prerequisites
- Step 1: Load the data from Amazon S3 to Amazon Redshift
- Step 2: Create the machine learning model
- Step 3: Perform predictions with the model

### Prerequisites

To complete this tutorial, you must complete the [Administrative setup](#) for Amazon Redshift ML.

### Step 1: Load the data from Amazon S3 to Amazon Redshift

Use the [Amazon Redshift query editor v2](#) to run the following queries. These queries load the sample data into Amazon Redshift.

1. The following query creates a table named ecommerce\_sales.

```
CREATE TABLE IF NOT EXISTS ecommerce_sales (
 invoice_no VARCHAR(30),
 stockcode VARCHAR(30),
 description VARCHAR(60),
 quantity DOUBLE PRECISION,
 invoice_date VARCHAR(30),
 unitprice DOUBLE PRECISION,
 customerid BIGINT,
 country VARCHAR(25)
);
```

2. The following query copies the sample data from the [E-Commerce Sales Forecast dataset](#) into the ecommerce\_sales table.

```
COPY ecommerce_sales
FROM
 's3://redshift-ml-multiclass/ecommerce_data.txt'
IAM_ROLE default
DELIMITER '\t'
IGNOREHEADER 1
REGION 'us-east-1'
MAXERROR 100;
```

### Split the data

When you create a model in Amazon Redshift ML, SageMaker automatically splits your data into training and test sets, so that SageMaker can determine the model accuracy. By manually splitting the data at this step, you will be able to verify the accuracy of the model by allocating an additional prediction set.

Use the following SQL statement to split the data into three sets for training, validation, and prediction.

```
--creates table with all data
CREATE TABLE ecommerce_sales_data AS (
 SELECT
 t1.stockcode,
 t1.description,
 t1.invoice_date,
 t1.customerid,
 t1.country,
 t1.sales_amt,
 CAST(RANDOM() * 100 AS INT) AS data_group_id
 FROM
 (
 SELECT
 stockcode,
 description,
 invoice_date,
 customerid,
 country,
 SUM(quantity * unitprice) AS sales_amt
 FROM
 ecommerce_sales
 GROUP BY
 1,
 2,
 3,
 4,
 5
) t1
);
```

```
--creates training set
CREATE TABLE ecommerce_sales_training AS (
 SELECT
 a.customerid,
 a.country,
 a.stockcode,
 a.description,
 a.invoicedate,
 a.sales_amt,
 (b.nbr_months_active) AS nbr_months_active
 FROM
 ecommerce_sales_data a
 INNER JOIN (
 SELECT
 customerid,
 COUNT(
 DISTINCT(
 DATE_PART(y, CAST(invoicedate AS DATE)) || '-' || LPAD(
 DATE_PART(mon, CAST(invoicedate AS DATE)),
 2,
 '00'
)
)
) AS nbr_months_active
 FROM
 ecommerce_sales_data
 GROUP BY
 1
) b ON a.customerid = b.customerid
 WHERE
 a.data_group_id < 80
);

--creates validation set
CREATE TABLE ecommerce_sales_validation AS (
 SELECT
 a.customerid,
 a.country,
 a.stockcode,
 a.description,
 a.invoicedate,
 a.sales_amt,
 (b.nbr_months_active) AS nbr_months_active
 FROM
 ecommerce_sales_data a
 INNER JOIN (
 SELECT
 customerid,
 COUNT(
 DISTINCT(
 DATE_PART(y, CAST(invoicedate AS DATE)) || '-' || LPAD(
 DATE_PART(mon, CAST(invoicedate AS DATE)),
 2,
 '00'
)
)
) AS nbr_months_active
 FROM
 ecommerce_sales_data
 GROUP BY
 1
) b ON a.customerid = b.customerid
 WHERE
 a.data_group_id BETWEEN 80
 AND 90
);
```

```
--creates prediction set
CREATE TABLE ecommerce_sales_prediction AS (
 SELECT
 customerid,
 country,
 stockcode,
 description,
 invoicedate,
 sales_amt
 FROM
 ecommerce_sales_data
 WHERE
 data_group_id > 90);
```

## Step 2: Create the machine learning model

In this step, you use the CREATE MODEL statement to create your machine learning model using multi-class classification.

The following query creates the multi-class classification model with the training set using the CREATE MODEL operation. Replace ***DOC-EXAMPLE-BUCKET*** with your own Amazon S3 bucket.

```
CREATE MODEL ecommerce_customer_activity
FROM
(
 SELECT
 customerid,
 country,
 stockcode,
 description,
 invoicedate,
 sales_amt,
 nbr_months_active
 FROM
 ecommerce_sales_training
) TARGET nbr_months_active FUNCTION predict_customer_activity IAM_ROLE default
PROBLEM_TYPE MULTICLASS_CLASSIFICATION SETTINGS (
 S3_BUCKET '<DOC-EXAMPLE-BUCKET>',
 S3_GARBAGE_COLLECT OFF
);
```

In this query, you specify the problem type as Multiclass\_Classification. The target that you predict for the model is nbr\_months\_active. When SageMaker finishes training the model, it creates the function predict\_customer\_activity, which you will use to make predictions in Amazon Redshift.

### Show the status of model training (optional)

You can use the SHOW MODEL command to know when your model is ready.

Use the following query to return various metrics of the model, including model state and accuracy.

```
SHOW MODEL ecommerce_customer_activity;
```

When the model is ready, the output of the previous operation should show that the Model State is Ready. The following is an example of the output of the SHOW MODEL operation.

```
+-----+
+-----+
+
```

|        |                                           |                                                     |                                 |
|--------|-------------------------------------------|-----------------------------------------------------|---------------------------------|
|        | Model Name                                |                                                     | ecommerce_customer_activity     |
| +----- |                                           |                                                     |                                 |
| +----- |                                           |                                                     |                                 |
|        | Schema Name                               |                                                     | public                          |
|        | Owner                                     |                                                     | awsuser                         |
|        | Creation Time                             |                                                     | Fri, 17.06.2022 19:02:15        |
|        | Model State                               |                                                     | READY                           |
|        | Training Job Status                       |                                                     | MaxAutoMLJobRuntimeReached      |
|        | validation:accuracy                       |                                                     | 0.991280                        |
|        | Estimated Cost                            |                                                     | 7.897689                        |
|        |                                           |                                                     |                                 |
|        | TRAINING DATA:                            |                                                     |                                 |
|        | Query                                     | SELECT CUSTOMERID, COUNTRY, STOCKCODE, DESCRIPTION, |                                 |
|        | INVOICEDATE, SALES_AMT, NBR_MONTHS_ACTIVE |                                                     | FROM ECOMMERCE_SALES_TRAINING   |
|        | Target Column                             |                                                     | NBR_MONTHS_ACTIVE               |
|        |                                           |                                                     |                                 |
|        | PARAMETERS:                               |                                                     |                                 |
|        | Model Type                                |                                                     | xgboost                         |
|        | Problem Type                              |                                                     | MulticlassClassification        |
|        | Objective                                 |                                                     | Accuracy                        |
|        | AutoML Job Name                           |                                                     | redshiftml-20220617190215268770 |
|        | Function Name                             |                                                     | predict_customer_activity       |
|        | Function Parameters                       | customerid country stockcode description            |                                 |
|        | invoicedate sales_amt                     |                                                     |                                 |
|        | Function Parameter Types                  | int8 varchar varchar varchar                        |                                 |
|        | float8                                    |                                                     |                                 |
|        | IAM Role                                  |                                                     | default-aws-iam-role            |
|        | S3 Bucket                                 |                                                     | DOC-EXAMPLE-BUCKET              |
|        | Max Runtime                               |                                                     | 5400                            |
| +----- |                                           |                                                     |                                 |
| +----- |                                           |                                                     |                                 |
| +      |                                           |                                                     |                                 |

### Step 3: Perform predictions with the model

The following query shows which customers qualify for your customer loyalty program. If the model predicts that the customer will be active for at least seven months, then the model selects the customer for the loyalty program.

```
SELECT
```

```

customerid,
predict_customer_activity(
 customerid,
 country,
 stockcode,
 description,
 invoicedate,
 sales_amt
) AS predicted_months_active
FROM
 ecommerce_sales_prediction
WHERE
 predicted_months_active >= 7
GROUP BY
 1,
 2
LIMIT
 10;

```

### [Run prediction queries against the validation data \(optional\)](#)

Run the following prediction queries against the validation data to see the model's level of accuracy.

```

SELECT
 CAST(SUM(t1.match) AS decimal(7, 2)) AS predicted_matches,
 CAST(SUM(t1.nonmatch) AS decimal(7, 2)) AS predicted_non_matches,
 CAST(SUM(t1.match + t1.nonmatch) AS decimal(7, 2)) AS total_predictions,
 predicted_matches / total_predictions AS pct_accuracy
FROM
(
 SELECT
 customerid,
 country,
 stockcode,
 description,
 invoicedate,
 sales_amt,
 nbr_months_active,
 predict_customer_activity(
 customerid,
 country,
 stockcode,
 description,
 invoicedate,
 sales_amt
) AS predicted_months_active,
 CASE
 WHEN nbr_months_active = predicted_months_active THEN 1
 ELSE 0
 END AS match,
 CASE
 WHEN nbr_months_active <> predicted_months_active THEN 1
 ELSE 0
 END AS nonmatch
 FROM
 ecommerce_sales_validation
) t1;

```

### [Predict how many customers miss entry \(optional\)](#)

The following query compares the number of customers that are predicted to be active for only 5 or 6 months. The model predicts that these customers will miss out on the loyalty program. The query then compares the amount that barely miss the program to the number that are predicted to be eligible for

the loyalty program. This query could be used to inform a decision on whether to lower the threshold for the loyalty program. You can also determine if there is a significant amount of customers that are predicted to barely miss out on the program. You could then encourage those customers to increase their activity to get a loyalty program membership.

```
SELECT
 predict_customer_activity(
 customerid,
 country,
 stockcode,
 description,
 invoicedate,
 sales_amt
) AS predicted_months_active,
 COUNT(customerid)
FROM
 ecommerce_sales_prediction
WHERE
 predicted_months_active BETWEEN 5 AND 6
GROUP BY
 1
ORDER BY
 1 ASC
LIMIT
 10)
UNION
(SELECT
 NULL AS predicted_months_active,
 COUNT (customerid)
FROM
 ecommerce_sales_prediction
WHERE
 predict_customer_activity(
 customerid,
 country,
 stockcode,
 description,
 invoicedate,
 sales_amt
) >=7);
```

## Related topics

For more information about Amazon Redshift ML, see the following documentation:

- [Costs for using Amazon Redshift ML](#)
- [CREATE MODEL operation](#)
- [EXPLAIN\\_MODEL function](#)

For more information about machine learning, see the following documentation:

- [Machine learning overview](#)
- [Machine learning for novices and experts](#)
- [What Is Fairness and Model Explainability for Machine Learning Predictions?](#)

## Tutorial: Building XGBoost models

In this tutorial, you create a model with data from Amazon S3 and run prediction queries with the model using Amazon Redshift ML. The XGBoost algorithm is an optimized implementation of the gradient

boosted trees algorithm. XGBoost handles more data types, relationships, and distributions than other gradient boosted trees algorithms. You can use XGBoost for regression, binary classification, multi-class classification, and ranking problems. For more information about the XGBoost algorithm, see [XGBoost algorithm](#) in the Amazon SageMaker Developer Guide.

The Amazon Redshift ML CREATE MODEL operation with the AUTO OFF option currently supports XGBoost as the MODEL\_TYPE. You can provide relevant information such as the objective and hyperparameters as part of the CREATE MODEL command, based on your use case.

In this tutorial, you use the [banknote authentication dataset](#), which is a binary classification problem to predict whether a given banknote is genuine or forged.

## Use case examples

You can solve other binary classification problems using Amazon Redshift ML, such as predicting whether a patient is healthy or has a disease. You could also predict whether an email is spam or not spam.

### Tasks

- Prerequisites
- Step 1: Load the data from Amazon S3 to Amazon Redshift
- Step 2: Create the machine learning model
- Step 3: Perform predictions with the model

## Prerequisites

To complete this tutorial, you must complete the [Administrative setup](#) for Amazon Redshift ML.

## Step 1: Load the data from Amazon S3 to Amazon Redshift

Use the [Amazon Redshift query editor v2](#) to run the following queries.

The following query creates two tables, loads the data from Amazon S3, and splits the data into a training set and a testing set. You will use the training set to train your model and create the prediction function. Then, you will test the prediction function on the testing set.

```
--create training set table
CREATE TABLE banknoteauthentication_train(
 variance FLOAT,
 skewness FLOAT,
 curtosis FLOAT,
 entropy FLOAT,
 class INT
);

--Load into training table
COPY banknoteauthentication_train
FROM
 's3://redshiftbucket-ml-sagemaker/banknote_authentication/train_data/' IAM_ROLE default
REGION 'us-west-2' IGNOREHEADER 1 CSV;

--create testing set table
CREATE TABLE banknoteauthentication_test(
 variance FLOAT,
 skewness FLOAT,
 curtosis FLOAT,
 entropy FLOAT,
 class INT
```

```

);
--Load data into testing table
COPY banknoteauthentication_test
FROM
's3://redshiftbucket-ml-sagemaker/banknote_authentication/test_data/'
IAM_ROLE default
REGION 'us-west-2'
IGNOREHEADER 1
CSV;

```

## Step 2: Create the machine learning model

The following query creates the XGBoost model in Amazon Redshift ML from the training set you created in the previous step. Replace `DOC-EXAMPLE-BUCKET` with your own `S3_BUCKET`, which will store your input datasets and other Redshift ML artifacts.

```

CREATE MODEL model_banknoteauthentication_xgboost_binary
FROM
 banknoteauthentication_train
TARGET class
FUNCTION func_model_banknoteauthentication_xgboost_binary
IAM_ROLE default
AUTO OFF
MODEL_TYPE xgboost
OBJECTIVE 'binary:logistic'
PREPROCESSORS 'none'
HYPERPARAMETERS DEFAULT
EXCEPT(NUM_ROUND '100')
SETTINGS(S3_BUCKET '<DOC-EXAMPLE-BUCKET>');

```

### Show the status of model training (optional)

You can use the `SHOW MODEL` command to know when your model is ready.

Use the following query to monitor the progress of the model training.

```
SHOW MODEL model_banknoteauthentication_xgboost_binary;
```

If the model is `READY`, the `SHOW MODEL` operation also provides the `train:error` metric, as shown in the following example of the output. The `train:error` metric is a measure of accuracy of your model that measures to six decimal places. A value of 0 is most accurate and a value of 1 is least accurate.

|                |                                                 |
|----------------|-------------------------------------------------|
| Model Name     | model_banknoteauthentication_xgboost_binary     |
| Schema Name    | public                                          |
| Owner          | awsuser                                         |
| Creation Time  | Tue, 21.06.2022 19:07:35                        |
| Model State    | READY                                           |
| train:error    | 0.000000                                        |
| Estimated Cost | 0.006197                                        |
| TRAINING DATA: |                                                 |
| Query          | SELECT *<br>FROM "BANKNOTEAUTHENTICATION_TRAIN" |
| Target Column  | CLASS                                           |
| PARAMETERS:    |                                                 |
| Model Type     | xgboost                                         |

|                          |                                                  |
|--------------------------|--------------------------------------------------|
| Training Job Name        | redshiftml-20220621190735686935-xgboost          |
| Function Name            | func_model_banknoteauthentication_xgboost_binary |
| Function Parameters      | variance skewness curtosis entropy               |
| Function Parameter Types | float8 float8 float8 float8                      |
| IAM Role                 | default-aws-iam-role                             |
| S3 Bucket                | DOC-EXAMPLE-BUCKET                               |
| Max Runtime              |                                                  |
|                          | 5400                                             |
| HYPERPARAMETERS:         |                                                  |
| num_round                |                                                  |
| objective                | 100                                              |
|                          | binary:logistic                                  |

## Step 3: Perform predictions with the model

### Check the accuracy of the model

The following prediction query uses the prediction function created in the previous step to check the accuracy of your model. Run this query on the testing set to make sure the model does not correspond too closely to the training set. This close correspondence is also known as overfitting, and overfitting could cause the model to make unreliable predictions.

```
WITH predict_data AS (
 SELECT
 class AS label,
 func_model_banknoteauthentication_xgboost_binary (variance, skewness, curtosis,
entropy) AS predicted,
 CASE
 WHEN label IS NULL THEN 0
 ELSE label
 END AS actual,
 CASE
 WHEN actual = predicted THEN 1 :: INT
 ELSE 0 :: INT
 END AS correct
 FROM
 banknoteauthentication_test
),
aggr_data AS (
 SELECT
 SUM(correct) AS num_correct,
 COUNT(*) AS total
 FROM
 predict_data
)
SELECT
 (num_correct :: FLOAT / total :: FLOAT) AS accuracy
FROM
 aggr_data;
```

### Predict the amount of original and counterfeit banknotes

The following prediction query returns the predicted amount of original and counterfeit banknotes in the testing set.

```
WITH predict_data AS (
 SELECT
 func_model_banknoteauthentication_xgboost_binary(variance, skewness, curtosis,
entropy) AS predicted
 FROM
```

```

 banknoteauthentication_test
)
SELECT
 CASE
 WHEN predicted = '0' THEN 'Original banknote'
 WHEN predicted = '1' THEN 'Counterfeit banknote'
 ELSE 'NA'
 END AS banknote_authentication,
 COUNT(1) AS count
FROM
 predict_data
GROUP BY
 1;

```

### Find the average observation for an original and a counterfeit banknote

The following prediction query returns the average value of each feature for banknotes that are predicted to be original and counterfeit in the testing set.

```

WITH predict_data AS (
 SELECT
 func_model_banknoteauthentication_xgboost_binary(variance, skewness, curtosis,
entropy) AS predicted,
 variance,
 skewness,
 curtosis,
 entropy
 FROM
 banknoteauthentication_test
)
SELECT
 CASE
 WHEN predicted = '0' THEN 'Original banknote'
 WHEN predicted = '1' THEN 'Counterfeit banknote'
 ELSE 'NA'
 END AS banknote_authentication,
 TRUNC(AVG(variance), 2) AS avg_variance,
 TRUNC(AVG(skewness), 2) AS avg_skewness,
 TRUNC(AVG(curtosis), 2) AS avg_curtosis,
 TRUNC(AVG(entropy), 2) AS avg_entropy
FROM
 predict_data
GROUP BY
 1
ORDER BY
 2;

```

### Related topics

For more information about Amazon Redshift ML, see the following documentation:

- [Costs for using Amazon Redshift ML](#)
- [CREATE MODEL operation](#)
- [EXPLAIN\\_MODEL function](#)

For more information about machine learning, see the following documentation:

- [Machine learning overview](#)
- [Machine learning for novices and experts](#)
- [What Is Fairness and Model Explainability for Machine Learning Predictions?](#)

## Tutorial: Building regression models

In this tutorial, you use Amazon Redshift ML to create a machine learning regression model and run prediction queries on the model. Regression models allow you to predict numerical outcomes, such as the price of a house, or how many people will use a city's bike rental service. You use the CREATE MODEL command in Amazon Redshift with your training data. Then, Amazon Redshift ML compiles the model, imports the trained model to Redshift, and prepares a SQL prediction function. You can use the prediction function in SQL queries in Amazon Redshift.

In this tutorial, you will use Amazon Redshift ML to build a regression model that predicts the number of people that use the city of Toronto's bike sharing service at any given hour of a day. The inputs for the model include holidays and weather conditions. You will use a regression model, because you want a numerical outcome for this problem.

You can use the CREATE MODEL command to export training data, train a model, and make the model available in Amazon Redshift as a SQL function. Use the CREATE MODEL operation to specify training data either as a table or a SELECT statement.

### Use case examples

You can solve other regression problems with Amazon Redshift ML, such as predicting a customer's lifetime value. You could also use Redshift ML to predict the most profitable price and the resulting revenue of a product.

### Tasks

- Prerequisites
- Step 1: Load the data from Amazon S3 to Amazon Redshift
- Step 2: Create the machine learning model
- Step 3: Validate the model

### Prerequisites

To complete this tutorial, you must complete the [Administrative setup](#) for Amazon Redshift ML.

### Step 1: Load the data from Amazon S3 to Amazon Redshift

Use the [Amazon Redshift query editor v2](#) to run the following queries.

1. You must create three tables to load the three public datasets into Amazon Redshift. The datasets are [Toronto Bike Ridership Data](#), [historical weather data](#), and [historical holidays data](#). Run the following query in the Amazon Redshift query editor to create tables named `ridership`, `weather`, and `holiday`.

```
CREATE TABLE IF NOT EXISTS ridership (
 trip_id INT,
 trip_duration_seconds INT,
 trip_start_time timestamp,
 trip_stop_time timestamp,
 from_station_name VARCHAR(50),
 to_station_name VARCHAR(50),
 from_station_id SMALLINT,
 to_station_id SMALLINT,
 user_type VARCHAR(20)
);

CREATE TABLE IF NOT EXISTS weather (
 longitude_x DECIMAL(5, 2),
```

```

latitude_y DECIMAL(5, 2),
station_name VARCHAR(20),
climate_id BIGINT,
datetime_utc TIMESTAMP,
weather_year SMALLINT,
weather_month SMALLINT,
weather_day SMALLINT,
time_utc VARCHAR(5),
temp_c DECIMAL(5, 2),
temp_flag VARCHAR(1),
dew_point_temp_c DECIMAL(5, 2),
dew_point_temp_flag VARCHAR(1),
rel_hum SMALLINT,
rel_hum_flag VARCHAR(1),
precip_amount_mm DECIMAL(5, 2),
precip_amount_flag VARCHAR(1),
wind_dir_10s_deg VARCHAR(10),
wind_dir_flag VARCHAR(1),
wind_spd_kmh VARCHAR(10),
wind_spd_flag VARCHAR(1),
visibility_km VARCHAR(10),
visibility_flag VARCHAR(1),
stn_press_kpa DECIMAL(5, 2),
stn_press_flag VARCHAR(1),
hmdx SMALLINT,
hmdx_flag VARCHAR(1),
wind_chill VARCHAR(10),
wind_chill_flag VARCHAR(1),
weather VARCHAR(10)
);

CREATE TABLE IF NOT EXISTS holiday (holiday_date DATE, description VARCHAR(100));

```

2. The following query loads the sample data into the tables that you created in the previous step.

```

COPY ridership
FROM
's3://redshift-ml-bikesharing-data/bike-sharing-data/ridership/'
IAM_ROLE default
FORMAT CSV
IGNOREHEADER 1
DATEFORMAT 'auto'
TIMEFORMAT 'auto'
REGION 'us-west-2'
gzip;

COPY weather
FROM
's3://redshift-ml-bikesharing-data/bike-sharing-data/weather/'
IAM_ROLE default
FORMAT csv
IGNOREHEADER 1
DATEFORMAT 'auto'
TIMEFORMAT 'auto'
REGION 'us-west-2'
gzip;

COPY holiday
FROM
's3://redshift-ml-bikesharing-data/bike-sharing-data/holiday/'
IAM_ROLE default
FORMAT csv
IGNOREHEADER 1
DATEFORMAT 'auto'
TIMEFORMAT 'auto'

```

```
REGION 'us-west-2'
gzip;
```

3. The following query performs transformations on the `ridership` and `weather` datasets to remove bias or anomalies. Removing bias and anomalies results in improved model accuracy. The query simplifies the tables by creating two new views called `ridership_view` and `weather_view`.

```
CREATE
OR REPLACE VIEW ridership_view AS
SELECT
 trip_time,
 trip_count,
 TO_CHAR(trip_time, 'hh24') :: INT trip_hour,
 TO_CHAR(trip_time, 'dd') :: INT trip_day,
 TO_CHAR(trip_time, 'mm') :: INT trip_month,
 TO_CHAR(trip_time, 'yy') :: INT trip_year,
 TO_CHAR(trip_time, 'q') :: INT trip_quarter,
 TO_CHAR(trip_time, 'w') :: INT trip_month_week,
 TO_CHAR(trip_time, 'd') :: INT trip_week_day
FROM
(
 SELECT
 CASE
 WHEN TRUNC(r.trip_start_time) < '2017-07-01' :: DATE THEN
 CONVERT_TZ(
 'US/Eastern',
 DATE_TRUNC('hour', r.trip_start_time)
)
 ELSE DATE_TRUNC('hour', r.trip_start_time)
 END trip_time,
 COUNT(1) trip_count
 FROM
 ridership r
 WHERE
 r.trip_duration_seconds BETWEEN 60
 AND 60 * 60 * 24
 GROUP BY
 1
);
CREATE
OR REPLACE VIEW weather_view AS
SELECT
 CONVERT_TZ(
 'US/Eastern',
 DATE_TRUNC('hour', datetime_utc)
) daytime,
 ROUND(AVG(temp_c)) temp_c,
 ROUND(AVG(precip_amount_mm)) precip_amount_mm
FROM
 weather
GROUP BY
 1;
```

4. The following query creates a table that combines all the relevant input attributes from `ridership_view` and `weather_view` into the `trip_data` table.

```
CREATE TABLE trip_data AS
SELECT
 r.trip_time,
 r.trip_count,
 r.trip_hour,
 r.trip_day,
 r.trip_month,
```

```

r.trip_year,
r.trip_quarter,
r.trip_month_week,
r.trip_week_day,
w.temp_c,
w.precip_amount_mm,CASE
 WHEN h.holiday_date IS NOT NULL THEN 1
 WHEN TO_CHAR(r.trip_time, 'D') :: INT IN (1, 7) THEN 1
 ELSE 0
END is_holiday,
ROW_NUMBER() OVER (
 ORDER BY
 RANDOM()
) serial_number
FROM
 ridership_view r
JOIN weather_view w ON (r.trip_time = w.daytime)
LEFT OUTER JOIN holiday h ON (TRUNC(r.trip_time) = h.holiday_date);

```

### [View the sample data \(optional\)](#)

The following query shows entries from the table. You can run this operation to make sure the table was made correctly.

```

SELECT *
FROM trip_data
LIMIT 5;

```

The following is an example of the output of the previous operation.

| trip_time           | trip_count | trip_hour | trip_day | trip_month | trip_year | trip_quarter | trip_month_week | trip_week_day | temp_c | precip_amount_mm | is_holiday | serial_number |
|---------------------|------------|-----------|----------|------------|-----------|--------------|-----------------|---------------|--------|------------------|------------|---------------|
| 2017-03-21 22:00:00 | 47         | 22        | 21       | 3          | 17        | 1            | 3               | 1             | 0      | 0                | 0          | 1             |
| 2018-05-04 01:00:00 | 19         | 1         | 4        | 5          | 18        | 2            | 1               | 12            | 0      | 0                | 0          | 3             |
| 2018-01-11 10:00:00 | 93         | 10        | 11       | 1          | 18        | 1            | 2               | 5             | 0      | 1                | 0          | 5             |
| 2017-10-28 04:00:00 | 20         | 4         | 28       | 10         | 17        | 4            | 4               | 7             | 0      | 10               | 1          | 7             |
| 2017-12-31 21:00:00 | 11         | 21        | 31       | 1          | 17        | 4            | 5               | 1             | -15    | 12               | 1          | 9             |

### Show the correlation between attributes (optional)

Determining correlation helps you measure the strength of association between attributes. The level of association can help you determine what affects your target output. In this tutorial, the target output is `trip_count`.

The following query creates or replaces the `sp_correlation` procedure. You use the stored procedure called `sp_correlation` to show the correlation between an attribute and other attributes in a table in Amazon Redshift.

```

CREATE OR REPLACE PROCEDURE sp_correlation(source_schema_name in varchar(255),
 source_table_name in varchar(255), target_column_name in varchar(255),
 output_temp_table_name inout varchar(255)) AS $$

DECLARE
 v_sql varchar(max);
 v_generated_sql varchar(max);
 v_source_schema_name varchar(255)=lower(source_schema_name);
 v_source_table_name varchar(255)=lower(source_table_name);
 v_target_column_name varchar(255)=lower(target_column_name);

BEGIN
 EXECUTE 'DROP TABLE IF EXISTS ' || output_temp_table_name;
 v_sql = '';
 SELECT
 '''CREATE temp table '''|| output_temp_table_name||' AS SELECT ''|| outer_calculation ||
 '' FROM (SELECT COUNT(1) number_of_items, SUM('||v_target_column_name||')
 sum_target, SUM(POW('||v_target_column_name||',2)) sum_square_target, POW(SUM('||
 v_target_column_name||'),2) square_sum_target,'||
 inner_calculation ||
 '' FROM (SELECT ''|||
 column_name ||
 '' FROM '|||v_source_table_name||'))'
 FROM
 (
 SELECT
 DISTINCT
 LISTAGG(outer_calculation,'','') OVER () outer_calculation
 ,LISTAGG(inner_calculation,'','') OVER () inner_calculation
 ,LISTAGG(column_name,'','') OVER () column_name
 FROM
 (
 SELECT
 CASE WHEN atttypid=16 THEN ''DECODE(''||column_name||',true,1,0)'' ELSE column_name
 END column_name
 ,atttypid
 ,''CAST(DECODE(number_of_items * sum_square_'''||rn||'' - square_sum_'''||rn||'',0,null,(number_of_items*sum_target_'''||rn||'' - sum_target * sum_'''||rn||'')/SQRT((number_of_items * sum_square_target - square_sum_target) * (number_of_items * sum_square_'''||rn||'' - square_sum_'''||rn||'))) AS numeric(5,2)) ''||column_name outer_calculation
 ,''sum(''||column_name||')' sum_'''||rn||'',''||
 ''SUM(trip_count'''||column_name||')' sum_target_'''||rn||'',''||
 ''SUM(POW(''||column_name||',2))' sum_square_'''||rn||'',''||
 ''POW(SUM(''||column_name||'),2)' square_sum_'''||rn|| inner_calculation
 FROM
 (
 SELECT
 row_number() OVER (order by a.attnum) rn
 ,a.attname::VARCHAR column_name
 ,a.atttypid
 FROM pg_namespace AS n
 INNER JOIN pg_class AS c ON n.oid = c.relnamespace
 INNER JOIN pg_attribute AS a ON c.oid = a.attrelid
 WHERE a.attnum > 0
 AND n.nspname = ''|||v_source_schema_name||''

```

```

 AND c.relname = '''||v_source_table_name||'''
 AND a.atttypid IN (16,20,21,23,700,701,1700)
)
)
);
EXECUTE v_sql INTO v_generated_sql;
EXECUTE v_generated_sql;
END;
$$ LANGUAGE plpgsql;

```

The following query shows the correlation between the target column, `trip_count`, and other numeric attributes in our dataset.

```

call sp_correlation(
 'public',
 'trip_data',
 'trip_count',
 'tmp_corr_table'
);

SELECT
 *
FROM
 tmp_corr_table;

```

The following example shows the output of the previous `sp_correlation` operation.

|   | trip_count | trip_hour | trip_day | trip_month | trip_year | trip_quarter | trip_month_week | trip_week_day | temp_c | precip_amount_mm | is_holiday | serial_number |
|---|------------|-----------|----------|------------|-----------|--------------|-----------------|---------------|--------|------------------|------------|---------------|
| 0 | 1          | 0.32      | 0.01     | 0.18       | 0.12      | 0.18         | 0.02            | 0.53          | -0.07  | -0.13            | 0          |               |

## Step 2: Create the machine learning model

1. The following query splits your data into a training set and a validation set by designating 80% of the dataset for training and 20% for validation. The training set is the input for the ML model to identify the best possible algorithm for the model. After the model is created, you use the validation set to validate the model accuracy.

```

CREATE TABLE training_data AS
SELECT
 trip_count,
 trip_hour,
 trip_day,
 trip_month,
 trip_year,
 trip_quarter,
 trip_month_week,
 trip_week_day,
 temp_c,
 precip_amount_mm,
 is_holiday

```

```

FROM
 trip_data
WHERE
 serial_number > (
 SELECT
 COUNT(1) * 0.2
 FROM
 trip_data
);

CREATE TABLE validation_data AS
SELECT
 trip_count,
 trip_hour,
 trip_day,
 trip_month,
 trip_year,
 trip_quarter,
 trip_month_week,
 trip_week_day,
 temp_c,
 precip_amount_mm,
 is_holiday,
 trip_time
FROM
 trip_data
WHERE
 serial_number <= (
 SELECT
 COUNT(1) * 0.2
 FROM
 trip_data
);

```

2. The following query creates a regression model to predict the `trip_count` value for any input date and time. In the following example, replace `DOC-EXAMPLE-BUCKET` with your own S3 bucket.

```

CREATE MODEL predict_rental_count
FROM
 training_data TARGET trip_count FUNCTION predict_rental_count
 IAM_ROLE default
 PROBLEM_TYPE regression
 OBJECTIVE 'mse'
 SETTINGS (
 s3_bucket '<DOC-EXAMPLE-BUCKET>',
 s3_garbage_collect off,
 max_runtime 5000
);

```

## Step 3: Validate the model

1. Use the following query to output aspects of the model, and find the mean square error metric in the output. Mean square error is a typical accuracy metric for regression problems.

```
show model predict_rental_count;
```

2. Run the following prediction queries against the validation data to compare the predicted trip count to the actual trip count.

```

SELECT
 trip_time,

```

```

 actual_count,
 predicted_count,
 (actual_count - predicted_count) difference
FROM
(
 SELECT
 trip_time,
 trip_count AS actual_count,
 PREDICT_RENTAL_COUNT (
 trip_hour,
 trip_day,
 trip_month,
 trip_year,
 trip_quarter,
 trip_month_week,
 trip_week_day,
 temp_c,
 precip_amount_mm,
 is_holiday
) predicted_count
 FROM
 validation_data
)
LIMIT
5;

```

3. The following query calculates the mean square error and root mean square error based on your validation data. You use mean square error and root mean square error to measure the distance between the predicted numeric target and the actual numeric answer. A good model has a low score in both metrics. The following query returns the value of both metrics.

```

SELECT
 ROUND(
 AVG(POWER((actual_count - predicted_count), 2)),
 2
) mse,
 ROUND(
 SQRT(AVG(POWER((actual_count - predicted_count), 2))),
 2
) rmse
FROM
(
 SELECT
 trip_time,
 trip_count AS actual_count,
 PREDICT_RENTAL_COUNT (
 trip_hour,
 trip_day,
 trip_month,
 trip_year,
 trip_quarter,
 trip_month_week,
 trip_week_day,
 temp_c,
 precip_amount_mm,
 is_holiday
) predicted_count
 FROM
 validation_data
);

```

4. The following query calculates the percent error in trip count for each trip time on 2017-01-01. The query orders the trip times from the time with the lowest percent error to the time with the highest percent error.

```
SELECT
 trip_time,
 CAST(ABS(((actual_count - predicted_count) / actual_count)) * 100 AS DECIMAL (7,2))
AS pct_error
FROM
(
 SELECT
 trip_time,
 trip_count AS actual_count,
 PREDICT_RENTAL_COUNT (
 trip_hour,
 trip_day,
 trip_month,
 trip_year,
 trip_quarter,
 trip_month_week,
 trip_week_day,
 temp_c,
 precip_amount_mm,
 is_holiday
) predicted_count
 FROM
 validation_data
)
WHERE
 trip_time LIKE '2017-01-01 %:%:%'
ORDER BY
 2 ASC;
```

## Related topics

For more information about Amazon Redshift ML, see the following documentation:

- [Costs for using Amazon Redshift ML](#)
- [CREATE MODEL operation](#)
- [EXPLAIN\\_MODEL function](#)

For more information about machine learning, see the following documentation:

- [Machine learning overview](#)
- [Machine learning for novices and experts](#)
- [What Is Fairness and Model Explainability for Machine Learning Predictions?](#)

## Tutorial: Building regression models with linear learner

In this tutorial, you create a linear learner model with data from Amazon S3 and run prediction queries with the model using Amazon Redshift ML. The SageMaker linear learner algorithm solves either regression or multi-class classification problems. To learn more about regression and multi-class classification problems, see [Problem types for the machine learning paradigms](#) in the Amazon SageMaker Developer Guide. In this tutorial, you solve a regression problem. The linear learner algorithm trains many models in parallel, and automatically determines the most optimized model. You use the CREATE MODEL operation in Amazon Redshift, which creates your linear learner model using SageMaker and sends a prediction function to Amazon Redshift. For more information about the linear learner algorithm, see [Linear Learner Algorithm](#) in the Amazon SageMaker Developer Guide.

You can use a CREATE MODEL command to export training data, train a model, import the model, and prepare an Amazon Redshift prediction function. Use the CREATE MODEL operation to specify training data either as a table or a SELECT statement.

Linear learner models optimize either continuous objectives or discrete objectives. Continuous objectives are used for regression, while discrete variables are used for classification. Some methods provide a solution for only continuous objectives, such as the regression method. The linear learner algorithm provides an increase in speed over naive hyperparameter optimization techniques, such as the Naive Bayes technique. A naive optimization technique assumes that each input variable is independent. To use the linear learner algorithm, you must provide columns representing the dimensions of the inputs, and rows representing the observations. For more information about the linear learner algorithm, see the [Linear Learner Algorithm](#) in the Amazon SageMaker Developer Guide.

In this tutorial, you build a linear learner model that predicts the age of abalone. You use the CREATE MODEL command on the [Abalone dataset](#) to determine the relationship between the physical measurements of abalone. Then, you use the model to determine the age of abalone.

## Use case examples

You can solve other regression problems with linear learner and Amazon Redshift ML, such as predicting the price of a house. You could also use Redshift ML to predict the number of people who will use a city's bike rental service.

### Tasks

- Prerequisites
- Step 1: Load the data from Amazon S3 to Amazon Redshift
- Step 2: Create the machine learning model
- Step 3: Validate the model

## Prerequisites

To complete this tutorial, you must complete the [Administrative setup](#) for Amazon Redshift ML.

## Step 1: Load the data from Amazon S3 to Amazon Redshift

Use the [Amazon Redshift query editor v2](#) to run the following queries. These queries load the sample data into Redshift and divide the data into a training set and a validation set.

1. The following query creates the abalone\_dataset table.

```
CREATE TABLE abalone_dataset (
 id INT IDENTITY(1, 1),
 Sex CHAR(1),
 Length float,
 Diameter float,
 Height float,
 Whole float,
 Shucked float,
 Viscera float,
 Shell float,
 Rings integer
);
```

2. The following query copies the sample data from the [Abalone dataset](#) in Amazon S3 to the abalone\_dataset table you created previously in Amazon Redshift.

```
COPY abalone_dataset
```

```
FROM
's3://redshift-ml-multiclass/abalone.csv' REGION 'us-east-1' IAM_ROLE default CSV
IGNOREHEADER 1 NULL AS 'NULL';
```

3. By manually splitting the data, you will be able to verify the accuracy of the model by allocating an additional prediction set. The following query splits the data into two sets. The abalone\_training table is for training and the abalone\_validation table is for validation.

```
CREATE TABLE abalone_training as
SELECT
*
FROM
 abalone_dataset
WHERE
 mod(id, 10) < 8;

CREATE TABLE abalone_validation as
SELECT
*
FROM
 abalone_dataset
WHERE
 mod(id, 10) >= 8;
```

## Step 2: Create the machine learning model

In this step, you use the CREATE MODEL statement to create your machine learning model with the linear learner algorithm.

The following query creates the linear learner model with the CREATE MODEL operation using your S3 bucket. Replace *DOC-EXAMPLE-BUCKET* with your own S3 bucket.

```
CREATE MODEL model_abalone_ring_prediction
FROM
(
 SELECT
 Sex,
 Length,
 Diameter,
 Height,
 Whole,
 Shucked,
 Viscera,
 Shell,
 Rings AS target_label
 FROM
 abalone_training
) TARGET target_label FUNCTION f_abalone_ring_prediction IAM_ROLE default MODEL_TYPE
LINEAR_LEARNER PROBLEM_TYPE REGRESSION OBJECTIVE 'MSE' SETTINGS (
 S3_BUCKET 'DOC-EXAMPLE-BUCKET',
 MAX_RUNTIME 15000
);
```

### Show the status of model training (optional)

You can use the SHOW MODEL command to know when your model is ready.

Use the following query to monitor the progress of the model training.

```
SHOW MODEL model_abalone_ring_prediction;
```

When the model is ready, the output of the previous operation should look similar to the following example. Note that the output provides the `validation:mse` metric, which is the mean square error. You will use the mean square error to validate the accuracy of the model in the next step.

|                                                                                                            |  |
|------------------------------------------------------------------------------------------------------------|--|
| +-----                                                                                                     |  |
| +-----                                                                                                     |  |
| +-----                                                                                                     |  |
| Model Name                                                                                                 |  |
| model_abalone_ring_prediction                                                                              |  |
| +-----                                                                                                     |  |
| +-----                                                                                                     |  |
| Schema Name   public                                                                                       |  |
| Owner   awsuser                                                                                            |  |
| Creation Time   Thu, 30.06.2022 18:00:10                                                                   |  |
| Model State   READY                                                                                        |  |
| validation:mse   4.168633                                                                                  |  |
| Estimated Cost   4.291608                                                                                  |  |
|                                                                                                            |  |
|                                                                                                            |  |
| TRAINING DATA:                                                                                             |  |
|                                                                                                            |  |
| Query   SELECT SEX , LENGTH , DIAMETER , HEIGHT , WHOLE , SHUCKED , VISCERA , SHELL, RINGS AS TARGET_LABEL |  |
| FROM ABALONE_TRAINING                                                                                      |  |
| Target Column   TARGET_LABEL                                                                               |  |
|                                                                                                            |  |
|                                                                                                            |  |
| PARAMETERS:                                                                                                |  |
|                                                                                                            |  |
| Model Type   linear_learner                                                                                |  |
| Problem Type   Regression                                                                                  |  |
| Objective   MSE                                                                                            |  |
| AutoML Job Name   redshiftml-20220630180010947843                                                          |  |
| Function Name   f_abalone_ring_prediction                                                                  |  |
| Function Parameters   sex length diameter height whole shucked viscera shell                               |  |
| Function Parameter Types   bpchar float8 float8 float8 float8 float8 float8 float8                         |  |
| IAM Role   default-aws-iam-role                                                                            |  |
| S3 Bucket   <b>DOC-EXAMPLE-BUCKET</b>                                                                      |  |
| Max Runtime   15000                                                                                        |  |
| +-----                                                                                                     |  |
| +-----                                                                                                     |  |
| +-----                                                                                                     |  |

## Step 3: Validate the model

1. The following prediction query validates the accuracy of the model on the abalone\_validation dataset by calculating mean square error and root mean square error.

```

SELECT
 ROUND(AVG(POWER((tgt_label - predicted), 2)), 2) mse,
 ROUND(SQRT(AVG(POWER((tgt_label - predicted), 2))), 2) rmse
FROM
(
 SELECT
 Sex,
 Length,
 Diameter,
 Height,
 Whole,
 Shucked,
 Viscera,
 Shell,
 Rings AS tgt_label,
 f_abalone_ring_prediction(
 Sex,
 Length,
 Diameter,
 Height,
 Whole,
 Shucked,
 Viscera,
 Shell
) AS predicted,
 CASE
 WHEN tgt_label = predicted then 1
 ELSE 0
 END AS match,
 CASE
 WHEN tgt_label <> predicted then 1
 ELSE 0
 END AS nonmatch
 FROM
 abalone_validation
) t1;

```

The output of the previous query should look like the following example. The value of the mean square error metric should be similar to the validation:mse metric shown by the SHOW MODEL operation's output.

| mse | rmse               |
|-----|--------------------|
| 5.1 | 2.2600000000000002 |

2. Use the following query to run the EXPLAIN\_MODEL operation on your prediction function. The operation will return a model explainability report. For more information about the EXPLAIN\_MODEL operation, see the [EXPLAIN\\_MODEL function](#) in the Amazon Redshift Database Developer Guide.

```

SELECT
 EXPLAIN_MODEL ('model_abalone_ring_prediction');

```

The following information is an example of the model explainability report produced by the previous EXPLAIN\_MODEL operation. The values for each of the inputs are Shapley values. The Shapley values

represent the effect each input has on the prediction of your model, with higher-valued inputs having more impact on the prediction. In this example, the higher-valued inputs have more impact on predicting the age of abalone.

```
{
 "explanations": {
 "kernel_shap": {
 "label0": {
 "expected_value" :10.290688514709473,
 "global_shap_values": {
 "diameter" :0.6856910187882492,
 "height" :0.4415323937124035,
 "length" :0.21507476107609084,
 "sex" :0.448611774505744,
 "shell" :1.70426496893776,
 "shucked" :2.1181392924386994,
 "viscera" :0.342220754059912,
 "whole" :0.6711906974084011
 }
 }
 }
 },
 "version" :"1.0"
};
```

3. Use the following query to calculate the percentage of correct predictions that the model makes about abalone that are not yet mature. Abalone that are immature have 10 rings or less, and a correct prediction is accurate to within one ring of the actual number of rings.

```
SELECT
 TRUNC(
 SUM(
 CASE
 WHEN ROUND(
 f_abalone_ring_prediction(
 Sex,
 Length,
 Diameter,
 Height,
 Whole,
 Shucked,
 Viscera,
 Shell
),
 0
) BETWEEN Rings - 1
 AND Rings + 1 THEN 1
 ELSE 0
 END
) / CAST(COUNT(SHELL) AS FLOAT),
 4
) AS prediction_pct
FROM
 abalone_validation
WHERE
 Rings <= 10;
```

## Related topics

For more information about Amazon Redshift ML, see the following documentation:

- [Costs for using Amazon Redshift ML](#)
- [CREATE MODEL operation](#)
- [EXPLAIN\\_MODEL function](#)

For more information about machine learning, see the following documentation:

- [Machine learning overview](#)
- [Machine learning for novices and experts](#)
- [What Is Fairness and Model Explainability for Machine Learning Predictions?](#)

## Tutorial: Building multi-class classification models with linear learner

In this tutorial, you create a linear learner model with data from Amazon S3, and then run prediction queries with the model using Amazon Redshift ML. The SageMaker linear learner algorithm solves either regression or classification problems. To learn more about regression and multi-class classification problems, see [Problem types for the machine learning paradigms](#) in the Amazon SageMaker Developer Guide. In this tutorial, you solve a multi-class classification problem. The linear learner algorithm trains many models in parallel, and automatically determines the most optimized model. You use the CREATE MODEL operation in Amazon Redshift, which creates your linear learner model using SageMaker and sends the prediction function to Amazon Redshift. For more information about the linear learner algorithm, see the [Linear Learner Algorithm](#) in the Amazon SageMaker Developer Guide.

You can use a CREATE MODEL command to export training data, train a model, import the model, and prepare an Amazon Redshift prediction function. Use the CREATE MODEL operation to specify training data either as a table or a SELECT statement.

Linear learner models optimize either continuous objectives or discrete objectives. Continuous objectives are used for regression, while discrete variables are used for classification. Some methods provide a solution for only continuous objectives, such as a regression method. The linear learner algorithm provides an increase in speed over naive hyperparameter optimization techniques, such as the Naive Bayes technique. A naive optimization technique assumes that each input variable is independent. The linear learner algorithm trains many models in parallel and selects the most optimized model. A similar algorithm is XGBoost, which combines estimates from a set of simpler and weaker models to make predictions. To learn more about XGBoost, see [XGBoost algorithm](#) in the Amazon SageMaker Developer Guide.

To use the linear learner algorithm, you must provide columns representing the dimensions of the inputs, and rows representing the observations. For more information about the linear learner algorithm, see the [Linear Learner Algorithm](#) in the Amazon SageMaker Developer Guide.

In this tutorial, you build a linear learner model that predicts the types of cover for a given area. You use the CREATE MODEL command on the [Covertype dataset](#) from the UCI Machine Learning Repository. Then, you use the prediction function created by the command to determine the types of cover in a wilderness area. A forest cover type is usually a type of tree. The inputs that Redshift ML will use to create the model include soil type, distance to roadways, and wilderness area designation. For more information about the dataset, see the [Covertype Dataset](#) from the UCI Machine Learning Repository.

### Use case examples

You can solve other multi-class classification problems with linear learner with Amazon Redshift ML, such as predicting the species of a plant from an image. You could also predict the quantity of a product that a customer will purchase.

## Tasks

- Prerequisites
- Step 1: Load the data from Amazon S3 to Amazon Redshift
- Step 2: Create the machine learning model
- Step 3: Validate the model

## Prerequisites

To complete this tutorial, you must complete the [Administrative setup](#) for Amazon Redshift ML.

### Step 1: Load the data from Amazon S3 to Amazon Redshift

Use the [Amazon Redshift query editor v2](#) to run the following queries. These queries load the sample data into Redshift and divide the data into a training set and a validation set.

1. The following query creates the `covertype_data` table.

```
CREATE TABLE public.covertype_data (
 elevation bigint ENCODE az64,
 aspect bigint ENCODE az64,
 slope bigint ENCODE az64,
 horizontal_distance_to_hydrology bigint ENCODE az64,
 vertical_distance_to_hydrology bigint ENCODE az64,
 horizontal_distance_to_roadways bigint ENCODE az64,
 hillshade_9am bigint ENCODE az64,
 hillshade_noon bigint ENCODE az64,
 hillshade_3pm bigint ENCODE az64,
 horizontal_distance_to_fire_points bigint ENCODE az64,
 wilderness_area1 bigint ENCODE az64,
 wilderness_area2 bigint ENCODE az64,
 wilderness_area3 bigint ENCODE az64,
 wilderness_area4 bigint ENCODE az64,
 soil_type1 bigint ENCODE az64,
 soil_type2 bigint ENCODE az64,
 soil_type3 bigint ENCODE az64,
 soil_type4 bigint ENCODE az64,
 soil_type5 bigint ENCODE az64,
 soil_type6 bigint ENCODE az64,
 soil_type7 bigint ENCODE az64,
 soil_type8 bigint ENCODE az64,
 soil_type9 bigint ENCODE az64,
 soil_type10 bigint ENCODE az64,
 soil_type11 bigint ENCODE az64,
 soil_type12 bigint ENCODE az64,
 soil_type13 bigint ENCODE az64,
 soil_type14 bigint ENCODE az64,
 soil_type15 bigint ENCODE az64,
 soil_type16 bigint ENCODE az64,
 soil_type17 bigint ENCODE az64,
 soil_type18 bigint ENCODE az64,
 soil_type19 bigint ENCODE az64,
 soil_type20 bigint ENCODE az64,
 soil_type21 bigint ENCODE az64,
 soil_type22 bigint ENCODE az64,
 soil_type23 bigint ENCODE az64,
 soil_type24 bigint ENCODE az64,
 soil_type25 bigint ENCODE az64,
 soil_type26 bigint ENCODE az64,
 soil_type27 bigint ENCODE az64,
 soil_type28 bigint ENCODE az64,
 soil_type29 bigint ENCODE az64,
```

```

soil_type30 bigint ENCODE az64,
soil_type31 bigint ENCODE az64,
soil_type32 bigint ENCODE az64,
soil_type33 bigint ENCODE az64,
soil_type34 bigint ENCODE az64,
soil_type35 bigint ENCODE az64,
soil_type36 bigint ENCODE az64,
soil_type37 bigint ENCODE az64,
soil_type38 bigint ENCODE az64,
soil_type39 bigint ENCODE az64,
soil_type40 bigint ENCODE az64,
cover_type bigint ENCODE az64
) DISTSTYLE AUTO;

```

2. The following query copies the sample data from the [Covertype dataset](#) in Amazon S3 to the covertype\_data table you created previously in Amazon Redshift.

```

COPY public.covertype_data
FROM
's3://redshift-ml-muliiclass/covtype.data.gz' IAM_ROLE DEFAULT gzip DELIMITER ','
REGION 'us-east-1';

```

3. By manually splitting the data, you will be able to verify the accuracy of the model by allocating an additional testing set. The following query splits the data into three sets. The covertype\_training table is for training, the covertype\_validation table is for validation, and the covertype\_test table is for testing your model. You will use the training set to train your model and the validation set to validate the model's development. Then, you use the testing set to test the performance of the model and see if the model is overfitting or underfitting the dataset.

```

CREATE TABLE public.covertype_data_prep AS
SELECT
 a.*,
 CAST (random() * 100 AS int) AS data_group_id
FROM
 public.covertype_data a;

--training dataset
CREATE TABLE public.covertype_training AS
SELECT
 *
FROM
 public.covertype_data_prep
WHERE
 data_group_id < 80;

--validation dataset
CREATE TABLE public.covertype_validation AS
SELECT
 *
FROM
 public.covertype_data_prep
WHERE
 data_group_id BETWEEN 80
 AND 89;

--test dataset
CREATE TABLE public.covertype_test AS
SELECT
 *
FROM
 public.covertype_data_prep
WHERE
 data_group_id > 89;

```

## Step 2: Create the machine learning model

In this step, you use the CREATE MODEL statement to create your machine learning model with the linear learner algorithm.

The following query creates the linear learner model with the CREATE MODEL operation using your S3 bucket. Replace *DOC-EXAMPLE-BUCKET* with your own S3 bucket.

```
CREATE MODEL forest_cover_type_model
FROM
(
 SELECT
 Elevation,
 Aspect,
 Slope,
 Horizontal_distance_to_hydrology,
 Vertical_distance_to_hydrology,
 Horizontal_distance_to_roadways,
 Hillshade_9am,
 Hillshade_noon,
 Hillshade_3pm,
 Horizontal_Distance_To_Fire_Points,
 Wilderness_Area1,
 Wilderness_Area2,
 Wilderness_Area3,
 Wilderness_Area4,
 soil_type1,
 Soil_Type2,
 Soil_Type3,
 Soil_Type4,
 Soil_Type5,
 Soil_Type6,
 Soil_Type7,
 Soil_Type8,
 Soil_Type9,
 Soil_Type10,
 Soil_Type11,
 Soil_Type12,
 Soil_Type13,
 Soil_Type14,
 Soil_Type15,
 Soil_Type16,
 Soil_Type17,
 Soil_Type18,
 Soil_Type19,
 Soil_Type20,
 Soil_Type21,
 Soil_Type22,
 Soil_Type23,
 Soil_Type24,
 Soil_Type25,
 Soil_Type26,
 Soil_Type27,
 Soil_Type28,
 Soil_Type29,
 Soil_Type30,
 Soil_Type31,
 Soil_Type32,
 Soil_Type33,
 Soil_Type34,
 Soil_Type36,
 Soil_Type37,
 Soil_Type38,
 Soil_Type39,
```

```
Soil_Type40,
Cover_type
from
public.covertype_training
) TARGET cover_type FUNCTION predict_cover_type IAM_ROLE default MODEL_TYPE
LINEAR_LEARNER PROBLEM_TYPE MULTICLASS_CLASSIFICATION OBJECTIVE 'Accuracy' SETTINGS (
S3_BUCKET '<DOC-EXAMPLE-BUCKET>',
S3_GARBAGE_COLLECT OFF,
MAX_RUNTIME 15000
);
```

### Show the status of model training (optional)

You can use the SHOW MODEL command to know when your model is ready.

Use the following query to monitor the progress of the model training.

```
SHOW MODEL forest_cover_type_model;
```

When the model is ready, the output of the previous operation should look similar to the following example. Note that the output provides the validation:multiclass\_accuracy metric, which you can view on the righthand side of the following example. Multi-class accuracy measures the percentage of data points that are classified correctly by the model. You will use multi-class accuracy to validate the accuracy of the model in the next step.

| Key         | Value                   |
|-------------|-------------------------|
| Model Name  | forest_cover_type_model |
| Schema Name | public                  |
| Owner       | awsuser                 |

|                                |                          |
|--------------------------------|--------------------------|
| Creation Time                  | Tue, 12.07.2022 20:24:32 |
| Model State                    | READY                    |
| validation:multiclass_accuracy |                          |
| Estimated Cost                 |                          |
| 0.724952                       |                          |
| 5.341750                       |                          |
| TRAINING DATA:                 |                          |

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Query         | SELECT ELEVATION, ASPECT, SLOPE,<br>HORIZONTAL_DISTANCE_TO_HYDROLOGY, VERTICAL_DISTANCE_TO_HYDROLOGY,<br>HORIZONTAL_DISTANCE_TO_ROADWAYS, HILLSHADE_9AM, HILLSHADE_NOON, HILLSHADE_3PM ,<br>HORIZONTAL_DISTANCE_TO_FIRE_POINTS, WILDERNESS_AREA1, WILDERNESS_AREA2, WILDERNESS_AREA3,<br>WILDERNESS_AREA4, SOIL_TYPE1, SOIL_TYPE2, SOIL_TYPE3, SOIL_TYPE4, SOIL_TYPE5, SOIL_TYPE6,<br>SOIL_TYPE7, SOIL_TYPE8, SOIL_TYPE9, SOIL_TYPE10 , SOIL_TYPE11, SOIL_TYPE12 , SOIL_TYPE13 ,<br>SOIL_TYPE14, SOIL_TYPE15, SOIL_TYPE16, SOIL_TYPE17, SOIL_TYPE18, SOIL_TYPE19, SOIL_TYPE20,<br>SOIL_TYPE21, SOIL_TYPE22, SOIL_TYPE23, SOIL_TYPE24, SOIL_TYPE25, SOIL_TYPE26, SOIL_TYPE27,<br>SOIL_TYPE28, SOIL_TYPE29, SOIL_TYPE30, SOIL_TYPE31, SOIL_TYPE32, SOIL_TYPE33, SOIL_TYPE34,<br>SOIL_TYPE36, SOIL_TYPE37, SOIL_TYPE38, SOIL_TYPE39, SOIL_TYPE40, COVER_TYPE  <br>  FROM PUBLIC.COVERTYPE_TRAINING |
| Target Column | COVER_TYPE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| PARAMETERS:   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Model Type    | linear_learner                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Problem Type  | MulticlassClassification                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

|                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Objective                | Accuracy                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| AutoML Job Name          | redshiftml-20220712202432187659                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Function Name            | predict_cover_type                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Function Parameters      | elevation aspect slope horizontal_distance_to_hydrology<br>vertical_distance_to_hydrology horizontal_distance_to_roadways hillshade_9am<br>hillshade_noon hillshade_3pm horizontal_distance_to_fire_points wilderness_area1<br>wilderness_area2 wilderness_area3 wilderness_area4 soil_type1 soil_type2 soil_type3<br>soil_type4 soil_type5 soil_type6 soil_type7 soil_type8 soil_type9 soil_type10 soil_type11<br>soil_type12 soil_type13 soil_type14 soil_type15 soil_type16 soil_type17 soil_type18<br>soil_type19 soil_type20 soil_type21 soil_type22 soil_type23 soil_type24 soil_type25<br>soil_type26 soil_type27 soil_type28 soil_type29 soil_type30 soil_type31 soil_type32<br>soil_type33 soil_type34 soil_type36 soil_type37 soil_type38 soil_type39 soil_type40 |
| Function Parameter Types | int8 int8<br>int8 int8 int8<br>int8 int8 int8<br>int8 int8                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| IAM Role                 | default-aws-iam-role                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

|             |                           |
|-------------|---------------------------|
| S3 Bucket   | <i>DOC-EXAMPLE-BUCKET</i> |
| Max Runtime |                           |
|             | 15000                     |
| +-----      |                           |
| +-----      |                           |
| +           |                           |

### Step 3: Validate the model

1. The following prediction query validates the accuracy of the model on the `covertype_validation` dataset by calculating multi-class accuracy. Multi-class accuracy is the percentage of the model's predictions that are correct.

```

SELECT
 CAST(sum(t1.match) AS decimal(7, 2)) AS predicted_matches,
 CAST(sum(t1.nonmatch) AS decimal(7, 2)) AS predicted_non_matches,
 CAST(sum(t1.match + t1.nonmatch) AS decimal(7, 2)) AS total_predictions,
 predicted_matches / total_predictions AS pct_accuracy
FROM
 (
 SELECT
 Elevation,
 Aspect,
 Slope,
 Horizontal_distance_to_hydrology,
 Vertical_distance_to_hydrology,
 Horizontal_distance_to_roadways,
 Hillshade_9am,
 Hillshade_noon,
 Hillshade_3pm,
 Horizontal_Distance_To_Fire_Points,
 Wilderness_Area1,
 Wilderness_Area2,
 Wilderness_Area3,
 Wilderness_Area4,
 soil_type1,
 Soil_Type2,
 Soil_Type3,
 Soil_Type4,
 Soil_Type5,
 Soil_Type6,
 Soil_Type7,
 Soil_Type8,
 Soil_Type9,
 Soil_Type10,
 Soil_Type11,
 Soil_Type12,

```

```
Soil_Type13,
Soil_Type14,
Soil_Type15,
Soil_Type16,
Soil_Type17,
Soil_Type18,
Soil_Type19,
Soil_Type20,
Soil_Type21,
Soil_Type22,
Soil_Type23,
Soil_Type24,
Soil_Type25,
Soil_Type26,
Soil_Type27,
Soil_Type28,
Soil_Type29,
Soil_Type30,
Soil_Type31,
Soil_Type32,
Soil_Type33,
Soil_Type34,
Soil_Type36,
Soil_Type37,
Soil_Type38,
Soil_Type39,
Soil_Type40,
Cover_type AS actual_cover_type,
predict_cover_type(
 Elevation,
 Aspect,
 Slope,
 Horizontal_distance_to_hydrology,
 Vertical_distance_to_hydrology,
 Horizontal_distance_to_roadways,
 Hillshade_9am,
 Hillshade_noon,
 Hillshade_3pm,
 Horizontal_Distance_To_Fire_Points,
 Wilderness_Area1,
 Wilderness_Area2,
 Wilderness_Area3,
 Wilderness_Area4,
 soil_type1,
 Soil_Type2,
 Soil_Type3,
 Soil_Type4,
 Soil_Type5,
 Soil_Type6,
 Soil_Type7,
 Soil_Type8,
 Soil_Type9,
 Soil_Type10,
 Soil_Type11,
 Soil_Type12,
 Soil_Type13,
 Soil_Type14,
 Soil_Type15,
 Soil_Type16,
 Soil_Type17,
 Soil_Type18,
 Soil_Type19,
 Soil_Type20,
 Soil_Type21,
 Soil_Type22,
 Soil_Type23,
```

```

Soil_Type24,
Soil_Type25,
Soil_Type26,
Soil_Type27,
Soil_Type28,
Soil_Type29,
Soil_Type30,
Soil_Type31,
Soil_Type32,
Soil_Type33,
Soil_Type34,
Soil_Type36,
Soil_Type37,
Soil_Type38,
Soil_Type39,
Soil_Type40
) AS predicted_cover_type,
CASE
 WHEN actual_cover_type = predicted_cover_type THEN 1
 ELSE 0
END AS match,
CASE
 WHEN actual_cover_type <> predicted_cover_type THEN 1
 ELSE 0
END AS nonmatch
FROM
 public.covertype_validation
) t1;

```

The output of the previous query should look like the following example. The value of the multi-class accuracy metric should be similar to the validation:multiclass\_accuracy metric shown by the SHOW MODEL operation's output.

| predicted_matches | predicted_non_matches | total_predictions | pct_accuracy |
|-------------------|-----------------------|-------------------|--------------|
| 41211             | 16324                 | 57535             | 0.71627704   |

2. The following query predicts the most common cover type for wilderness\_area2. This dataset includes four wilderness areas and seven cover types. A wilderness area can have multiple cover types.

```

SELECT t1. predicted_cover_type, COUNT(*)
FROM
(
SELECT
 Elevation,
 Aspect,
 Slope,
 Horizontal_distance_to_hydrology,
 Vertical_distance_to_hydrology,
 Horizontal_distance_to_roadways,
 Hillshade_9am,
 Hillshade_noon,
 Hillshade_3pm ,
 Horizontal_Distance_To_Fire_Points,
 Wilderness_Area1,
 Wilderness_Area2,
 Wilderness_Area3,
 Wilderness_Area4,
 soil_type1,
 Soil_Type2,
 Soil_Type3,

```

```
Soil_Type4,
Soil_Type5,
Soil_Type6,
Soil_Type7,
Soil_Type8,
Soil_Type9,
Soil_Type10 ,
Soil_Type11,
Soil_Type12 ,
Soil_Type13 ,
Soil_Type14,
Soil_Type15,
Soil_Type16,
Soil_Type17,
Soil_Type18,
Soil_Type19,
Soil_Type20,
Soil_Type21,
Soil_Type22,
Soil_Type23,
Soil_Type24,
Soil_Type25,
Soil_Type26,
Soil_Type27,
Soil_Type28,
Soil_Type29,
Soil_Type30,
Soil_Type31,
Soil_Type32,
Soil_Type33,
Soil_Type34,
Soil_Type36,
Soil_Type37,
Soil_Type38,
Soil_Type39,
Soil_Type40,
predict_cover_type(Elevation,
Aspect,
Slope,
Horizontal_distance_to_hydrology,
Vertical_distance_to_hydrology,
Horizontal_distance_to_roadways,
Hillshade_9am,
Hillshade_noon,
Hillshade_3pm ,
Horizontal_Distance_To_Fire_Points,
Wilderness_Area1,
Wilderness_Area2,
Wilderness_Area3,
Wilderness_Area4,
soil_type1,
Soil_Type2,
Soil_Type3,
Soil_Type4,
Soil_Type5,
Soil_Type6,
Soil_Type7,
Soil_Type8,
Soil_Type9,
Soil_Type10,
Soil_Type11,
Soil_Type12,
Soil_Type13,
Soil_Type14,
Soil_Type15,
Soil_Type16,
```

```

Soil_Type17,
Soil_Type18,
Soil_Type19,
Soil_Type20,
Soil_Type21,
Soil_Type22,
Soil_Type23,
Soil_Type24,
Soil_Type25,
Soil_Type26,
Soil_Type27,
Soil_Type28,
Soil_Type29,
Soil_Type30,
Soil_Type31,
Soil_Type32,
Soil_Type33,
Soil_Type34,
Soil_Type36,
Soil_Type37,
Soil_Type38,
Soil_Type39,
Soil_Type40) AS predicted_cover_type

FROM public.covertype_test
WHERE wilderness_area2 = 1)
t1
GROUP BY 1;

```

The output of the previous operation should look similar to the following example. This output means that the model predicted that the majority of cover is cover type 1, and there is some cover of cover types 2 and 7.

| predicted_cover_type | count |
|----------------------|-------|
| 2                    | 564   |
| 7                    | 97    |
| 1                    | 2309  |

3. The following query shows the most common cover type in a single wilderness area. The query displays the amount of that cover type and the cover type's wilderness area.

```

SELECT t1. predicted_cover_type, COUNT(*), wilderness_area
FROM
(
SELECT
 Elevation,
 Aspect,
 Slope,
 Horizontal_distance_to_hydrology,
 Vertical_distance_to_hydrology,
 Horizontal_distance_to_roadways,
 Hillshade_9am,
 Hillshade_noon,
 Hillshade_3pm ,
 Horizontal_Distance_To_Fire_Points,
 Wilderness_Area1,
 Wilderness_Area2,
 Wilderness_Area3,
 Wilderness_Area4,
 soil_type1,

```

```
Soil_Type2,
Soil_Type3,
Soil_Type4,
Soil_Type5,
Soil_Type6,
Soil_Type7,
Soil_Type8,
Soil_Type9,
Soil_Type10 ,
Soil_Type11,
Soil_Type12 ,
Soil_Type13 ,
Soil_Type14,
Soil_Type15,
Soil_Type16,
Soil_Type17,
Soil_Type18,
Soil_Type19,
Soil_Type20,
Soil_Type21,
Soil_Type22,
Soil_Type23,
Soil_Type24,
Soil_Type25,
Soil_Type26,
Soil_Type27,
Soil_Type28,
Soil_Type29,
Soil_Type30,
Soil_Type31,
Soil_Type32,
Soil_Type33,
Soil_Type34,
Soil_Type36,
Soil_Type37,
Soil_Type38,
Soil_Type39,
Soil_Type40,
predict_cover_type(Elevation,
Aspect,
Slope,
Horizontal_distance_to_hydrology,
Vertical_distance_to_hydrology,
Horizontal_distance_to_roadways,
Hillshade_9am,
Hillshade_noon,
Hillshade_3pm ,
Horizontal_Distance_To_Fire_Points,
Wilderness_Area1,
Wilderness_Area2,
Wilderness_Area3,
Wilderness_Area4,
soil_type1,
Soil_Type2,
Soil_Type3,
Soil_Type4,
Soil_Type5,
Soil_Type6,
Soil_Type7,
Soil_Type8,
Soil_Type9,
Soil_Type10,
Soil_Type11,
Soil_Type12,
Soil_Type13,
Soil_Type14,
```

```
Soil_Type15,
Soil_Type16,
Soil_Type17,
Soil_Type18,
Soil_Type19,
Soil_Type20,
Soil_Type21,
Soil_Type22,
Soil_Type23,
Soil_Type24,
Soil_Type25,
Soil_Type26,
Soil_Type27,
Soil_Type28,
Soil_Type29,
Soil_Type30,
Soil_Type31,
Soil_Type32,
Soil_Type33,
Soil_Type34,
Soil_Type36,
Soil_Type37,
Soil_Type38,
Soil_Type39,
Soil_Type40) AS predicted_cover_type,
CASE WHEN Wilderness_Area1 = 1 THEN 1
WHEN Wilderness_Area2 = 1 THEN 2
WHEN Wilderness_Area3 = 1 THEN 3
WHEN Wilderness_Area4 = 1 THEN 4
ELSE 0
END AS wilderness_area

FROM public.covertype_test)
t1
GROUP BY 1, 3
ORDER BY 2 DESC
LIMIT 1;
```

The output of the previous operation should look similar to the following example.

| predicted_cover_type | count | wilderness_area |
|----------------------|-------|-----------------|
| 2                    | 15738 | 1               |

## Related topics

For more information about Amazon Redshift ML, see the following documentation:

- [Costs for using Amazon Redshift ML](#)
- [CREATE MODEL operation](#)
- [EXPLAIN\\_MODEL function](#)

For more information about machine learning, see the following documentation:

- [Machine learning overview](#)
- [Machine learning for novices and experts](#)
- [What Is Fairness and Model Explainability for Machine Learning Predictions?](#)

# Tuning query performance

Amazon Redshift uses queries based on structured query language (SQL) to interact with data and objects in the system. Data manipulation language (DML) is the subset of SQL that you use to view, add, change, and delete data. Data definition language (DDL) is the subset of SQL that you use to add, change, and delete database objects such as tables and views.

Once your system is set up, you typically work with DML the most, especially the [SELECT \(p. 928\)](#) command for retrieving and viewing data. To write effective data retrieval queries in Amazon Redshift, become familiar with SELECT and apply the tips outlined in [Amazon Redshift best practices for designing tables \(p. 15\)](#) to maximize query efficiency.

To understand how Amazon Redshift processes queries, use the [Query processing \(p. 463\)](#) and [Analyzing and improving queries \(p. 473\)](#) sections. Then you can apply this information in combination with diagnostic tools to identify and remove issues in query performance.

To identify and address some of the most common and most serious issues you are likely to encounter with Amazon Redshift queries, use the [Troubleshooting queries \(p. 486\)](#) section.

## Topics

- [Query processing \(p. 463\)](#)
- [Analyzing and improving queries \(p. 473\)](#)
- [Troubleshooting queries \(p. 486\)](#)

## Query processing

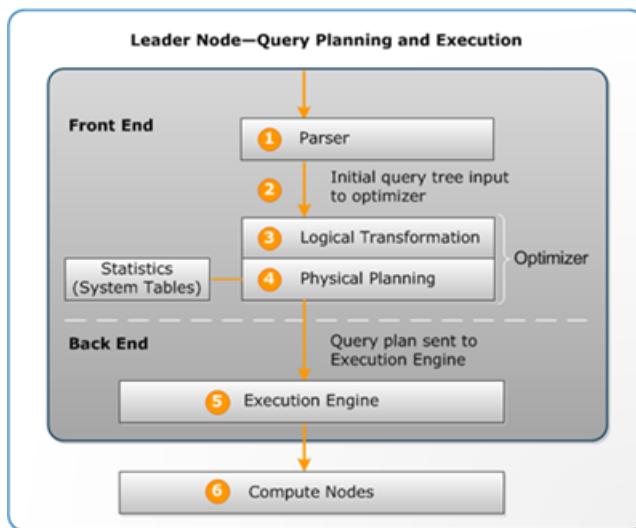
Amazon Redshift routes a submitted SQL query through the parser and optimizer to develop a query plan. The execution engine then translates the query plan into code and sends that code to the compute nodes for execution.

## Topics

- [Query planning and execution workflow \(p. 463\)](#)
- [Query plan \(p. 465\)](#)
- [Reviewing query plan steps \(p. 470\)](#)
- [Factors affecting query performance \(p. 472\)](#)

## Query planning and execution workflow

The following illustration provides a high-level view of the query planning and execution workflow.



The query planning and execution workflow follow these steps:

1. The leader node receives the query and parses the SQL.
2. The parser produces an initial query tree that is a logical representation of the original query. Amazon Redshift then inputs this query tree into the query optimizer.
3. The optimizer evaluates and if necessary rewrites the query to maximize its efficiency. This process sometimes results in creating multiple related queries to replace a single one.
4. The optimizer generates a query plan (or several, if the previous step resulted in multiple queries) for the execution with the best performance. The query plan specifies execution options such as join types, join order, aggregation options, and data distribution requirements.

You can use the [EXPLAIN \(p. 881\)](#) command to view the query plan. The query plan is a fundamental tool for analyzing and tuning complex queries. For more information, see [Query plan \(p. 465\)](#).

5. The execution engine translates the query plan into *steps*, *segments*, and *streams*:

#### Step

Each step is an individual operation needed during query execution. Steps can be combined to allow compute nodes to perform a query, join, or other database operation.

#### Segment

A combination of several steps that can be done by a single process, also the smallest compilation unit executable by a compute node slice. A *slice* is the unit of parallel processing in Amazon Redshift. The segments in a stream run in parallel.

#### Stream

A collection of segments to be parceled out over the available compute node slices.

The execution engine generates compiled code based on steps, segments, and streams. Compiled code runs faster than interpreted code and uses less compute capacity. This compiled code is then broadcast to the compute nodes.

#### Note

When benchmarking your queries, you should always compare the times for the second execution of a query, because the first execution time includes the overhead of compiling the code. For more information, see [Factors affecting query performance \(p. 472\)](#).

6. The compute node slices run the query segments in parallel. As part of this process, Amazon Redshift takes advantage of optimized network communication, memory, and disk management to pass intermediate results from one query plan step to the next. This also helps to speed query execution.

Steps 5 and 6 happen once for each stream. The engine creates the executable segments for one stream and sends them to the compute nodes. When the segments of that stream are complete, the engine generates the segments for the next stream. In this way, the engine can analyze what happened in the prior stream (for example, whether operations were disk-based) to influence the generation of segments in the next stream.

When the compute nodes are done, they return the query results to the leader node for final processing. The leader node merges the data into a single result set and addresses any needed sorting or aggregation. The leader node then returns the results to the client.

**Note**

The compute nodes might return some data to the leader node during query execution if necessary. For example, if you have a subquery with a LIMIT clause, the limit is applied on the leader node before data is redistributed across the cluster for further processing.

## Query plan

You can use the query plan to get information on the individual operations required to run a query. Before you work with a query plan, we recommend that you first understand how Amazon Redshift handles processing queries and creating query plans. For more information, see [Query planning and execution workflow \(p. 463\)](#).

To create a query plan, run the [EXPLAIN \(p. 881\)](#) command followed by the actual query text. The query plan gives you the following information:

- What operations the execution engine performs, reading the results from bottom to top.
- What type of step each operation performs.
- Which tables and columns are used in each operation.
- How much data is processed in each operation, in terms of number of rows and data width in bytes.
- The relative cost of the operation. Cost is a measure that compares the relative execution times of the steps within a plan. Cost does not provide any precise information about actual execution times or memory consumption, nor does it provide a meaningful comparison between execution plans. It does give you an indication of which operations in a query are consuming the most resources.

The EXPLAIN command doesn't actually run the query. It only shows the plan that Amazon Redshift runs if the query is run under current operating conditions. If you change the schema or data for a table and run [ANALYZE \(p. 663\)](#) again to update the statistical metadata, the query plan might be different.

The query plan output by EXPLAIN is a simplified, high-level view of query execution. It doesn't illustrate the details of parallel query processing. To see detailed information, run the query itself, and then get query summary information from the SVL\_QUERY\_SUMMARY or SVL\_QUERY\_REPORT view. For more information about using these views, see [Analyzing the query summary \(p. 476\)](#).

The following example shows the EXPLAIN output for a simple GROUP BY query on the EVENT table:

```
explain select eventname, count(*) from event group by eventname;

 QUERY PLAN

XN HashAggregate (cost=131.97..133.41 rows=576 width=17)
 -> XN Seq Scan on event (cost=0.00..87.98 rows=8798 width=17)
```

EXPLAIN returns the following metrics for each operation:

#### Cost

A relative value that is useful for comparing operations within a plan. Cost consists of two decimal values separated by two periods, for example cost=131.97..133.41. The first value, in this case 131.97, provides the relative cost of returning the first row for this operation. The second value, in this case 133.41, provides the relative cost of completing the operation. The costs in the query plan are cumulative as you read up the plan, so the HashAggregate cost in this example (131.97..133.41) includes the cost of the Seq Scan below it (0.00..87.98).

#### Rows

The estimated number of rows to return. In this example, the scan is expected to return 8798 rows. The HashAggregate operator on its own is expected to return 576 rows (after duplicate event names are discarded from the result set).

#### Note

The rows estimate is based on the available statistics generated by the ANALYZE command. If ANALYZE has not been run recently, the estimate is less reliable.

#### Width

The estimated width of the average row, in bytes. In this example, the average row is expected to be 17 bytes wide.

## EXPLAIN operators

This section briefly describes the operators that you see most often in the EXPLAIN output. For a complete list of operators, see [EXPLAIN \(p. 881\)](#) in the SQL Commands section.

### Sequential scan operator

The sequential scan operator (Seq Scan) indicates a table scan. Seq Scan scans each column in the table sequentially from beginning to end and evaluates query constraints (in the WHERE clause) for every row.

### Join operators

Amazon Redshift selects join operators based on the physical design of the tables being joined, the location of the data required for the join, and the specific requirements of the query itself.

- **Nested Loop**

The least optimal join, a nested loop is used mainly for cross-joins (Cartesian products) and some inequality joins.

- **Hash Join and Hash**

Typically faster than a nested loop join, a hash join and hash are used for inner joins and left and right outer joins. These operators are used when joining tables where the join columns are not both distribution keys *and* sort keys. The hash operator creates the hash table for the inner table in the join; the hash join operator reads the outer table, hashes the joining column, and finds matches in the inner hash table.

- **Merge Join**

Typically the fastest join, a merge join is used for inner joins and outer joins. The merge join is not used for full joins. This operator is used when joining tables where the join columns are both distribution keys *and* sort keys, and when less than 20 percent of the joining tables are unsorted. It reads two sorted tables in order and finds the matching rows. To view the percent of unsorted rows, query the [SVV\\_TABLE\\_INFO \(p. 1457\)](#) system table.

## Aggregate operators

The query plan uses the following operators in queries that involve aggregate functions and GROUP BY operations.

- **Aggregate**

Operator for scalar aggregate functions such as AVG and SUM.

- **HashAggregate**

Operator for unsorted grouped aggregate functions.

- **GroupAggregate**

Operator for sorted grouped aggregate functions.

## Sort operators

The query plan uses the following operators when queries have to sort or merge result sets.

- **Sort**

Evaluates the ORDER BY clause and other sort operations, such as sorts required by UNION queries and joins, SELECT DISTINCT queries, and window functions.

- **Merge**

Produces final sorted results according to intermediate sorted results that derive from parallel operations.

## UNION, INTERSECT, and EXCEPT operators

The query plan uses the following operators for queries that involve set operations with UNION, INTERSECT, and EXCEPT.

- **Subquery**

Used to run UNION queries.

- **Hash Intersect Distinct**

Used to run INTERSECT queries.

- **SetOp Except**

Used to run EXCEPT (or MINUS) queries.

## Other operators

The following operators also appear frequently in EXPLAIN output for routine queries.

- **Unique**

Removes duplicates for SELECT DISTINCT queries and UNION queries.

- **Limit**

Processes the LIMIT clause.

- **Window**

Runs window functions.

- **Result**

Runs scalar functions that do not involve any table access.

- **Subplan**

Used for certain subqueries.

- **Network**

Sends intermediate results to the leader node for further processing.

- **Materialize**

Saves rows for input to nested loop joins and some merge joins.

## Joins in EXPLAIN

The query optimizer uses different join types to retrieve table data, depending on the structure of the query and the underlying tables. The EXPLAIN output references the join type, the tables used, and the way the table data is distributed across the cluster to describe how the query is processed.

### Join type examples

The following examples show the different join types that the query optimizer can use. The join type used in the query plan depends on the physical design of the tables involved.

#### Example: Hash join two tables

The following query joins EVENT and CATEGORY on the CATID column. CATID is the distribution and sort key for CATEGORY but not for EVENT. A hash join is performed with EVENT as the outer table and CATEGORY as the inner table. Because CATEGORY is the smaller table, the planner broadcasts a copy of it to the compute nodes during query processing by using DS\_BCAST\_INNER. The join cost in this example accounts for most of the cumulative cost of the plan.

```
explain select * from category, event where category.catid=event.catid;
 QUERY PLAN

XN Hash Join DS_BCAST_INNER (cost=0.14..6600286.07 rows=8798 width=84)
 Hash Cond: ("outer".catid = "inner".catid)
 -> XN Seq Scan on event (cost=0.00..87.98 rows=8798 width=35)
 -> XN Hash (cost=0.11..0.11 rows=11 width=49)
 -> XN Seq Scan on category (cost=0.00..0.11 rows=11 width=49)
```

#### Note

Aligned indents for operators in the EXPLAIN output sometimes indicate that those operations do not depend on each other and can start in parallel. In the preceding example, although the scan on the EVENT table and the hash operation are aligned, the EVENT scan must wait until the hash operation has fully completed.

#### Example: Merge join two tables

The following query also uses SELECT \*, but it joins SALES and LISTING on the LISTID column, where LISTID has been set as both the distribution and sort key for both tables. A merge join is chosen, and no redistribution of data is required for the join (DS\_DIST\_NONE).

```
explain select * from sales, listing where sales.listid = listing.listid;
 QUERY PLAN

XN Merge Join DS_DIST_NONE (cost=0.00..6285.93 rows=172456 width=97)
 Merge Cond: ("outer".listid = "inner".listid)
```

```

-> XN Seq Scan on listing (cost=0.00..1924.97 rows=192497 width=44)
-> XN Seq Scan on sales (cost=0.00..1724.56 rows=172456 width=53)
```

The following example demonstrates the different types of joins within the same query. As in the previous example, SALES and LISTING are merge joined, but the third table, EVENT, must be hash joined with the results of the merge join. Again, the hash join incurs a broadcast cost.

```

explain select * from sales, listing, event
where sales.listid = listing.listid and sales.eventid = event.eventid;
 QUERY PLAN

XN Hash Join DS_BCAST_INNER (cost=109.98..3871130276.17 rows=172456 width=132)
 Hash Cond: ("outer".eventid = "inner".eventid)
 -> XN Merge Join DS_DIST_NONE (cost=0.00..6285.93 rows=172456 width=97)
 Merge Cond: ("outer".listid = "inner".listid)
 -> XN Seq Scan on listing (cost=0.00..1924.97 rows=192497 width=44)
 -> XN Seq Scan on sales (cost=0.00..1724.56 rows=172456 width=53)
 -> XN Hash (cost=87.98..87.98 rows=8798 width=35)
 -> XN Seq Scan on event (cost=0.00..87.98 rows=8798 width=35)
```

### Example: Join, aggregate, and sort

The following query runs a hash join of the SALES and EVENT tables, followed by aggregation and sort operations to account for the grouped SUM function and the ORDER BY clause. The initial sort operator runs in parallel on the compute nodes. Then the Network operator sends the results to the leader node, where the Merge operator produces the final sorted results.

```

explain select eventname, sum(pricepaid) from sales, event
where sales.eventid=event.eventid group by eventname
order by 2 desc;
 QUERY PLAN

XN Merge (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
 Merge Key: sum(sales.pricepaid)
 -> XN Network (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
 Send to leader
 -> XN Sort (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
 Sort Key: sum(sales.pricepaid)
 -> XN HashAggregate (cost=2815366577.07..2815366578.51 rows=576 width=27)
 -> XN Hash Join DS_BCAST_INNER (cost=109.98..2815365714.80
rows=172456 width=27)
 Hash Cond: ("outer".eventid = "inner".eventid)
 -> XN Seq Scan on sales (cost=0.00..1724.56 rows=172456
width=14)
 -> XN Hash (cost=87.98..87.98 rows=8798 width=21)
 -> XN Seq Scan on event (cost=0.00..87.98 rows=8798
width=21)
```

### Data redistribution

The EXPLAIN output for joins also specifies a method for how data is moved around a cluster to facilitate the join. This data movement can be either a broadcast or a redistribution. In a broadcast, the data values from one side of a join are copied from each compute node to every other compute node, so that every compute node ends up with a complete copy of the data. In a redistribution, participating data values are sent from their current slice to a new slice (possibly on a different node). Data is typically redistributed to match the distribution key of the other table participating in the join if that distribution key is one of the joining columns. If neither of the tables has distribution keys on one of the joining columns, either both tables are distributed or the inner table is broadcast to every node.

The EXPLAIN output also references inner and outer tables. The inner table is scanned first, and appears nearer the bottom of the query plan. The inner table is the table that is probed for matches. It is usually

held in memory, is usually the source table for hashing, and if possible, is the smaller table of the two being joined. The outer table is the source of rows to match against the inner table. It is usually read from disk. The query optimizer chooses the inner and outer table based on database statistics from the latest run of the ANALYZE command. The order of tables in the FROM clause of a query doesn't determine which table is inner and which is outer.

Use the following attributes in query plans to identify how data is moved to facilitate a query:

- **DS\_BCAST\_INNER**

A copy of the entire inner table is broadcast to all compute nodes.

- **DS\_DIST\_ALL\_NONE**

No redistribution is required, because the inner table has already been distributed to every node using DISTSTYLE ALL.

- **DS\_DIST\_NONE**

No tables are redistributed. Collocated joins are possible because corresponding slices are joined without moving data between nodes.

- **DS\_DIST\_INNER**

The inner table is redistributed.

- **DS\_DIST\_OUTER**

The outer table is redistributed.

- **DS\_DIST\_ALL\_INNER**

The entire inner table is redistributed to a single slice because the outer table uses DISTSTYLE ALL.

- **DS\_DIST\_BOTH**

Both tables are redistributed.

## Reviewing query plan steps

You can see the steps in a query plan by running the EXPLAIN command. The following example shows an SQL query and explains the output. Reading the query plan from the bottom up, you can see each of the logical operations used to perform the query. For more information, see [Query plan \(p. 465\)](#).

```
explain
select eventname, sum(pricepaid) from sales, event
where sales.eventid = event.eventid
group by eventname
order by 2 desc;
```

```
XN Merge (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
 Merge Key: sum(sales.pricepaid)
 -> XN Network (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
 Send to leader
 -> XN Sort (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
 Sort Key: sum(sales.pricepaid)
 -> XN HashAggregate (cost=2815366577.07..2815366578.51 rows=576 width=27)
 -> XN Hash Join DS_BCAST_INNER (cost=109.98..2815365714.80
rows=172456 width=27)
 Hash Cond: ("outer".eventid = "inner".eventid)
 -> XN Seq Scan on sales (cost=0.00..1724.56 rows=172456
width=14)
 -> XN Hash (cost=87.98..87.98 rows=8798 width=21)
```

-> XN Seq Scan on event (cost=0.00..87.98 rows=8798)

As part of generating a query plan, the query optimizer breaks down the plan into streams, segments, and steps. The query optimizer breaks the plan down to prepare for distributing the data and query workload to the compute nodes. For more information about streams, segments, and steps, see [Query planning and execution workflow \(p. 463\)](#).

The following illustration shows the preceding query and associated query plan. It displays how the query operations involved map to steps that Amazon Redshift uses to generate compiled code for the compute node slices. Each query plan operation maps to multiple steps within the segments, and sometimes to multiple segments within the streams.



In this illustration, the query optimizer runs the query plan as follows:

1. In Stream 0, the query runs Segment 0 with a sequential scan operation to scan the events table. The query continues to Segment 1 with a hash operation to create the hash table for the inner table in the join.
2. In Stream 1, the query runs Segment 2 with a sequential scan operation to scan the sales table. It continues with Segment 3 with a hash join to join tables where the join columns are not

both distribution keys and sort keys. It again continues with Segment 2 with a hash aggregate to aggregate results. Then the query runs Segment 3 with a hash aggregate operation to perform unsorted grouped aggregate functions, and a sort operation to evaluate the ORDER BY clause and other sort operations.

3. In Stream 2, the query runs a network operation in Segment 4 and Segment 5 to send intermediate results to the leader node for further processing.

The last segment of a query returns the data. If the return set is aggregated or sorted, the compute nodes each send their piece of the intermediate result to the leader node. The leader node then merges the data so the final result can be sent back to the requesting client.

For more information about EXPLAIN operators, see [EXPLAIN \(p. 881\)](#).

## Factors affecting query performance

A number of factors can affect query performance. The following aspects of your data, cluster, and database operations all play a part in how quickly your queries process.

- **Number of nodes, processors, or slices** – A compute node is partitioned into slices. More nodes means more processors and more slices, which enables your queries to process faster by running portions of the query concurrently across the slices. However, more nodes also means greater expense, so you need to find the balance of cost and performance that is appropriate for your system. For more information on Amazon Redshift cluster architecture, see [Data warehouse system architecture \(p. 3\)](#).
- **Node types** – An Amazon Redshift cluster can use one of several node types. Each node type offers different sizes and limits to help you scale your cluster appropriately. The node size determines the storage capacity, memory, CPU, and price of each node in the cluster. For more information about node types, see [Overview of Amazon Redshift clusters](#) in the *Amazon Redshift Management Guide*.
- **Data distribution** – Amazon Redshift stores table data on the compute nodes according to a table's distribution style. When you run a query, the query optimizer redistributes the data to the compute nodes as needed to perform any joins and aggregations. Choosing the right distribution style for a table helps minimize the impact of the redistribution step by locating the data where it needs to be before the joins are performed. For more information, see [Working with data distribution styles \(p. 49\)](#).
- **Data sort order** – Amazon Redshift stores table data on disk in sorted order according to a table's sort keys. The query optimizer and the query processor use the information about where the data is located to reduce the number of blocks that need to be scanned and thereby improve query speed. For more information, see [Working with sort keys \(p. 60\)](#).
- **Dataset size** – A higher volume of data in the cluster can slow query performance for queries, because more rows need to be scanned and redistributed. You can mitigate this effect by regular vacuuming and archiving of data, and by using a predicate to restrict the query dataset.
- **Concurrent operations** – Running multiple operations at once can affect query performance. Each operation takes one or more slots in an available query queue and uses the memory associated with those slots. If other operations are running, enough query queue slots might not be available. In this case, the query has to wait for slots to open before it can begin processing. For more information about creating and configuring query queues, see [Implementing workload management \(p. 491\)](#).
- **Query structure** – How your query is written affects its performance. As much as possible, write queries to process and return as little data as meets your needs. For more information, see [Amazon Redshift best practices for designing queries \(p. 21\)](#).
- **Code compilation** – Amazon Redshift generates and compiles code for each query execution plan.

The compiled code runs faster because it removes the overhead of using an interpreter. You generally have some overhead cost the first time code is generated and compiled. As a result, the performance of a query the first time you run it can be misleading. The overhead cost might be especially noticeable when you run one-off queries. Run the query a second time to determine its typical performance. Amazon Redshift uses a serverless compilation service to scale query compilations beyond the compute resources of an Amazon Redshift cluster. The compiled code segments are cached locally on

the cluster and in a virtually unlimited cache. This cache persists after cluster reboots. Subsequent executions of the same query run faster because they can skip the compilation phase. The cache is not compatible across Amazon Redshift versions, so the code is recompiled when queries run after a version upgrade. By using a scalable compilation service, Amazon Redshift can compile code in parallel to provide consistently fast performance. The magnitude of workload speed-up depends on the complexity and concurrency of queries.

## Analyzing and improving queries

Retrieving information from an Amazon Redshift data warehouse involves running complex queries on extremely large amounts of data, which can take a long time to process. To make sure that queries process as quickly as possible, there are a number of tools you can use to identify potential performance issues.

### Topics

- [Query analysis workflow \(p. 473\)](#)
- [Reviewing query alerts \(p. 474\)](#)
- [Analyzing the query plan \(p. 475\)](#)
- [Analyzing the query summary \(p. 476\)](#)
- [Improving query performance \(p. 481\)](#)
- [Diagnostic queries for query tuning \(p. 483\)](#)

## Query analysis workflow

If a query is taking longer than expected, use the following steps to identify and correct issues that might be negatively affecting the query's performance. If you aren't sure what queries in your system might benefit from performance tuning, start by running the diagnostic query in [Identifying queries that are top candidates for tuning \(p. 484\)](#).

1. Make sure that your tables are designed according to best practices. For more information, see [Amazon Redshift best practices for designing tables \(p. 15\)](#).
2. See if you can delete or archive any unneeded data in your tables. For example, suppose your queries always target the last 6 months' worth of data but you have the last 18 months' worth in your tables. In this case, you can delete or archive the older data to reduce the number of records that must be scanned and distributed.
3. Run the [VACUUM \(p. 1015\)](#) command on the tables in the query to reclaim space and re-sort rows. Running VACUUM helps if the unsorted region is large and the query uses the sort key in a join or in the predicate.
4. Run the [ANALYZE \(p. 663\)](#) command on the tables in the query to make sure that statistics are up to date. Running ANALYZE helps if any of the tables in the query have recently changed a lot in size. If running a full ANALYZE command will take too long, run ANALYZE on a single column to reduce processing time. This approach still updates the table size statistics; table size is a significant factor in query planning.
5. Make sure that your query has been run once for each type of client (based on what type of connection protocol the client uses) so that the query is compiled and cached. This approach speeds up subsequent runs of the query. For more information, see [Factors affecting query performance \(p. 472\)](#).
6. Check the [STL\\_ALERT\\_EVENT\\_LOG \(p. 1498\)](#) table to identify and correct possible issues with your query. For more information, see [Reviewing query alerts \(p. 474\)](#).
7. Run the [EXPLAIN \(p. 881\)](#) command to get the query plan and use it to optimize the query. For more information, see [Analyzing the query plan \(p. 475\)](#).

8. Use the [SVL\\_QUERY\\_SUMMARY \(p. 1647\)](#) and [SVL\\_QUERY\\_REPORT \(p. 1645\)](#) views to get summary information and use it to optimize the query. For more information, see [Analyzing the query summary \(p. 476\)](#).

Sometimes a query that should run quickly is forced to wait until another, longer-running query finishes. In that case, you might have nothing to improve in the query itself, but you can improve overall system performance by creating and using query queues for different types of queries. To get an idea of queue wait time for your queries, see [Reviewing queue wait times for queries \(p. 485\)](#). For more information about configuring query queues, see [Implementing workload management \(p. 491\)](#).

## Reviewing query alerts

To use the [STL\\_ALERT\\_EVENT\\_LOG \(p. 1498\)](#) system table to identify and correct potential performance issues with your query, follow these steps:

1. Run the following to determine your query ID:

```
select query, elapsed, substring
from svl_qlog
order by query
desc limit 5;
```

Examine the truncated query text in the `substring` field to determine which query value to select. If you have run the query more than once, use the query value from the row with the lower `elapsed` value. That is the row for the compiled version. If you have been running many queries, you can raise the value used by the `LIMIT` clause used to make sure that your query is included.

2. Select rows from `STL_ALERT_EVENT_LOG` for your query:

```
Select * from stl_alert_event_log where query = MyQueryID;
```

| userid | query  | slice | segment | step | pid  | xid    | event                                                                                  | solution            | event_time |
|--------|--------|-------|---------|------|------|--------|----------------------------------------------------------------------------------------|---------------------|------------|
| 100    | 32359  | 4     | 0       | 0    | 8780 | 71195  | Very selective query filter:ratio=rows(2)/r Review the choice of sort key to enable... | 2015-02-10 17:40:50 |            |
| 100    | 32359  | 5     | 0       | 0    | 8781 | 71195  | Very selective query filter:ratio=rows(2)/r Review the choice of sort key to enable... | 2015-02-10 17:40:50 |            |
| 100    | 109142 | 4     | 0       | 0    | 8780 | 302411 | Very selective query filter:ratio=rows(2)/r Review the choice of sort key to enable... | 2015-02-24 20:32:28 |            |
| 100    | 109142 | 5     | 0       | 0    | 8781 | 302411 | Very selective query filter:ratio=rows(2)/r Review the choice of sort key to enable... | 2015-02-24 20:32:28 |            |
| 100    | 109828 | 4     | 1       | 0    | 8746 | 304543 | Very selective query filter:ratio=rows(3)/r Review the choice of sort key to enable... | 2015-02-24 23:27:52 |            |
| 100    | 109828 | 5     | 1       | 0    | 8747 | 304543 | Very selective query filter:ratio=rows(3)/r Review the choice of sort key to enable... | 2015-02-24 23:27:52 |            |
| 100    | 109829 | 4     | 1       | 0    | 8760 | 304543 | Very selective query filter:ratio=rows(3)/r Review the choice of sort key to enable... | 2015-02-24 23:28:01 |            |
| 100    | 109829 | 5     | 1       | 0    | 8761 | 304543 | Very selective query filter:ratio=rows(3)/r Review the choice of sort key to enable... | 2015-02-24 23:28:01 |            |
| 100    | 113910 | 4     | 1       | 0    | 8774 | 316848 | Very selective query filter:ratio=rows(3)/r Review the choice of sort key to enable... | 2015-02-25 17:14:58 |            |
| 100    | 113910 | 5     | 1       | 0    | 8775 | 316848 | Very selective query filter:ratio=rows(3)/r Review the choice of sort key to enable... | 2015-02-25 17:14:58 |            |

3. Evaluate the results for your query. Use the following table to locate potential solutions for any issues that you have identified.

### Note

Not all queries have rows in `STL_ALERT_EVENT_LOG`, only those with identified issues.

| Issue                                                                   | Event value                        | Solution value                | Recommended solution                                                   |
|-------------------------------------------------------------------------|------------------------------------|-------------------------------|------------------------------------------------------------------------|
| Statistics for the tables in the query are missing or out of date.      | Missing query planner statistics   | Run the ANALYZE command       | See <a href="#">Table statistics missing or out of date (p. 481)</a> . |
| There is a nested loop join (the least optimal join) in the query plan. | Nested Loop Join in the query plan | Review the join predicates to | See <a href="#">Nested loop (p. 481)</a> .                             |

| Issue                                                                                                                                                                      | Event value                                                                                                               | Solution value                                                                                    | Recommended solution                                          |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|---------------------------------------------------------------|
|                                                                                                                                                                            |                                                                                                                           | avoid Cartesian products                                                                          |                                                               |
| The scan skipped a relatively large number of rows that are marked as deleted but not vacuumed, or rows that have been inserted but not committed.                         | Scanned a large number of deleted rows                                                                                    | Run the VACUUM command to reclaim deleted space                                                   | See <a href="#">Ghost rows or uncommitted rows (p. 482)</a> . |
| More than 1,000,000 rows were redistributed for a hash join or aggregation.                                                                                                | Distributed a large number of rows across the network: RowCount rows were distributed in order to process the aggregation | Review the choice of distribution key to collocate the join or aggregation                        | See <a href="#">Suboptimal data distribution (p. 482)</a> .   |
| More than 1,000,000 rows were broadcast for a hash join.                                                                                                                   | Broadcasted a large number of rows across the network                                                                     | Review the choice of distribution key to collocate the join and consider using distributed tables | See <a href="#">Suboptimal data distribution (p. 482)</a> .   |
| A DS_DIST_ALL_INNER redistribution style was indicated in the query plan, which forces serial execution because the entire inner table was redistributed to a single node. | DS_DIST_ALL_INNER for Hash Join in the query plan                                                                         | Review the choice of distribution strategy to distribute the inner, rather than outer, table      | See <a href="#">Suboptimal data distribution (p. 482)</a> .   |

## Analyzing the query plan

Before analyzing the query plan, you should be familiar with how to read it. If you are unfamiliar with reading a query plan, we recommend that you read [Query plan \(p. 465\)](#) before proceeding.

Run the [EXPLAIN \(p. 881\)](#) command to get a query plan. To analyze the data provided by the query plan, follow these steps:

1. Identify the steps with the highest cost. Concentrate on optimizing those when proceeding through the remaining steps.
2. Look at the join types:
  - **Nested Loop:** Such joins usually occur because a join condition was omitted. For recommended solutions, see [Nested loop \(p. 481\)](#).
  - **Hash and Hash Join:** Hash joins are used when joining tables where the join columns are not distribution keys and also not sort keys. For recommended solutions, see [Hash join \(p. 481\)](#).
  - **Merge Join:** No change is needed.
3. Notice which table is used for the inner join, and which for the outer join. The query engine generally chooses the smaller table for the inner join, and the larger table for the outer join. If such a choice doesn't occur, your statistics are likely out of date. For recommended solutions, see [Table statistics missing or out of date \(p. 481\)](#).

4. See if there are any high-cost sort operations. If there are, see [Unsorted or missorted rows \(p. 482\)](#) for recommended solutions.
5. Look for the following broadcast operators where there are high-cost operations:
  - **DS\_BCAST\_INNER**: Indicates that the table is broadcast to all the compute nodes. This is fine for a small table, but not ideal for a larger table.
  - **DS\_DIST\_ALL\_INNER**: Indicates that all of the workload is on a single slice.
  - **DS\_DIST\_BOTH**: Indicates heavy redistribution.

For recommended solutions for these situations, see [Suboptimal data distribution \(p. 482\)](#).

## Analyzing the query summary

To get execution steps and statistics in more detail than in the query plan that [EXPLAIN \(p. 881\)](#) produces, use the [SVL\\_QUERY\\_SUMMARY \(p. 1647\)](#) and [SVL\\_QUERY\\_REPORT \(p. 1645\)](#) system views.

`SVL_QUERY_SUMMARY` provides query statistics by stream. You can use the information it provides to identify issues with expensive steps, long-running steps, and steps that write to disk.

The `SVL_QUERY_REPORT` system view enables you to see information similar to that for `SVL_QUERY_SUMMARY`, only by compute node slice rather than by stream. You can use the slice-level information for detecting uneven data distribution across the cluster (also known as data distribution skew), which forces some nodes to do more work than others and impairs query performance.

### Topics

- [Using the SVL\\_QUERY\\_SUMMARY view \(p. 476\)](#)
- [Using the SVL\\_QUERY\\_REPORT view \(p. 478\)](#)
- [Mapping the query plan to the query summary \(p. 479\)](#)

## Using the `SVL_QUERY_SUMMARY` view

To analyze query summary information by stream, do the following:

1. Run the following query to determine your query ID:

```
select query, elapsed, substring
from svl_qlog
order by query
desc limit 5;
```

Examine the truncated query text in the `substring` field to determine which query value represents your query. If you have run the query more than once, use the query value from the row with the lower `elapsed` value. That is the row for the compiled version. If you have been running many queries, you can raise the value used by the `LIMIT` clause used to make sure that your query is included.

2. Select rows from `SVL_QUERY_SUMMARY` for your query. Order the results by stream, segment, and step:

```
select * from svl_query_summary where query = MyQueryID order by stm, seg, step;
```

| userid | query  | stm | seg | step | maxtime | avgtime | rows | bytes | rate_row | rate_byte | label                                  | is_diskbased | workmem     | is_rrscan | is_delayed_scan | rows_pre_filter |
|--------|--------|-----|-----|------|---------|---------|------|-------|----------|-----------|----------------------------------------|--------------|-------------|-----------|-----------------|-----------------|
| 1      | 249059 | 0   | 0   | 0    | 58      | 27      | 4    | 192   |          |           | scan tbl=246 name=Internal Worktable   | f            | 0 f         | f         |                 | 0               |
| 1      | 249059 | 0   | 0   | 1    | 58      | 27      | 4    | 0     |          |           | project                                | f            | 0 f         | f         |                 | 0               |
| 1      | 249059 | 0   | 0   | 2    | 58      | 27      | 4    | 64    |          |           | save tbl=249                           | f            | 481296384 f | f         |                 | 0               |
| 1      | 249059 | 1   | 1   | 0    | 20      | 20      | 1    | 48    |          |           | scan tbl=250 name=Internal Worktable   | f            | 0 f         | f         |                 | 0               |
| 1      | 249059 | 1   | 1   | 1    | 20      | 20      | 1    | 0     |          |           | dist                                   | f            | 0 f         | f         |                 | 0               |
| 1      | 249059 | 1   | 2   | 0    | 2275    | 1350    | 1    | 48    |          |           | scan tbl=19221 name=Internal Worktable | f            | 0 f         | f         |                 | 0               |
| 1      | 249059 | 1   | 2   | 1    | 2275    | 1350    | 1    | 0     |          |           | project                                | f            | 0 f         | f         |                 | 0               |
| 1      | 249059 | 1   | 2   | 2    | 2275    | 1350    | 1    | 16    |          |           | save tbl=249                           | f            | 475004928 f | f         |                 | 0               |
| 1      | 249059 | 2   | 3   | 0    | 1640    | 792     | 5    | 80    |          |           | scan tbl=249 name=Internal Worktable   | f            | 0 f         | f         |                 | 0               |
| 1      | 249059 | 2   | 3   | 1    | 1640    | 792     | 5    | 80    |          |           | sort tbl=248                           | f            | 468713472 f | f         |                 | 0               |
| 1      | 249059 | 3   | 4   | 0    | 26      | 9       | 5    | 80    |          |           | scan tbl=248 name=Internal Worktable   | f            | 0 f         | f         |                 | 0               |
| 1      | 249059 | 3   | 4   | 1    | 26      | 9       | 5    | 0     |          |           | return                                 | f            | 0 f         | f         |                 | 0               |
| 1      | 249059 | 3   | 5   | 0    | 49      | 49      | 0    | 0     |          |           | merge                                  | f            | 0 f         | f         |                 | 0               |
| 1      | 249059 | 3   | 5   | 1    | 49      | 49      | 5    | 0     |          |           | project                                | f            | 0 f         | f         |                 | 0               |
| 1      | 249059 | 3   | 5   | 2    | 49      | 49      | 0    | 0     |          |           | return                                 | f            | 0 f         | f         |                 | 0               |

- Map the steps to the operations in the query plan using the information in [Mapping the query plan to the query summary \(p. 479\)](#). They should have approximately the same values for rows and bytes (rows \* width from the query plan). If they don't, see [Table statistics missing or out of date \(p. 481\)](#) for recommended solutions.
- See if the is\_diskbased field has a value of t (true) for any step. Hashes, aggregates, and sorts are the operators that are likely to write data to disk if the system doesn't have enough memory allocated for query processing.

If is\_diskbased is true, see [Insufficient memory allocated to the query \(p. 483\)](#) for recommended solutions.

- Review the label field values and see if there is an AGG-DIST-AGG sequence anywhere in the steps. Its presence indicates two-step aggregation, which is expensive. To fix this, change the GROUP BY clause to use the distribution key (the first key, if there are multiple ones).
- Review the maxtime value for each segment (it is the same across all steps in the segment). Identify the segment with the highest maxtime value and review the steps in this segment for the following operators.

#### Note

A high maxtime value doesn't necessarily indicate a problem with the segment. Despite a high value, the segment might not have taken a long time to process. All segments in a stream start getting timed in unison. However, some downstream segments might not be able to run until they get data from upstream ones. This effect might make them seem to have taken a long time because their maxtime value includes both their waiting time and their processing time.

- BCAST or DIST:** In these cases, the high maxtime value might be the result of redistributing a large number of rows. For recommended solutions, see [Suboptimal data distribution \(p. 482\)](#).
- HJOIN (hash join):** If the step in question has a very high value in the rows field compared to the rows value in the final RETURN step in the query, see [Hash join \(p. 481\)](#) for recommended solutions.
- SCAN/SORT:** Look for a SCAN, SORT, SCAN, MERGE sequence of steps just before a join step. This pattern indicates that unsorted data is being scanned, sorted, and then merged with the sorted area of the table.

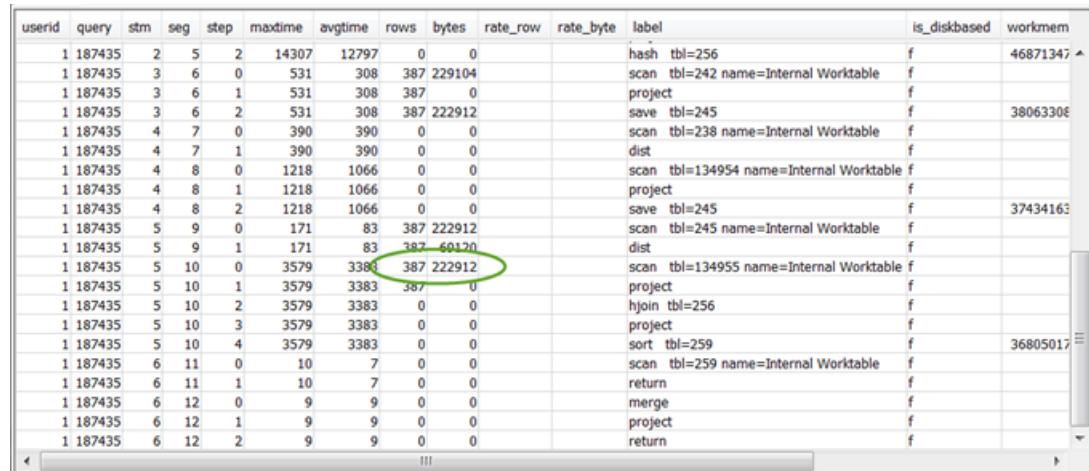
See if the rows value for the SCAN step has a very high value compared to the rows value in the final RETURN step in the query. This pattern indicates that the execution engine is scanning rows that are later discarded, which is inefficient. For recommended solutions, see [Insufficiently restrictive predicate \(p. 483\)](#).

If the maxtime value for the SCAN step is high, see [Suboptimal WHERE clause \(p. 483\)](#) for recommended solutions.

If the rows value for the SORT step is not zero, see [Unsorted or missorted rows \(p. 482\)](#) for recommended solutions.

- Review the rows and bytes values for the 5–10 steps that precede the final RETURN step to get an idea of the amount of data that is returned to the client. This process can be a bit of an art.

For example, in the following query summary, you can see that the third PROJECT step provides a rows value but not a bytes value. By looking through the preceding steps for one with the same rows value, you find the SCAN step that provides both rows and bytes information:



| userid | query  | stm | seg | step | maxtime | avgtime | rows | bytes  | rate_row | rate_byte | label                                   | is_diskbased | workmem    |
|--------|--------|-----|-----|------|---------|---------|------|--------|----------|-----------|-----------------------------------------|--------------|------------|
| 1      | 187435 | 2   | 5   | 2    | 14307   | 12797   | 0    | 0      |          |           | hash tbl=256                            | f            | 46871347 ▲ |
| 1      | 187435 | 3   | 6   | 0    | 531     | 308     | 387  | 229104 |          |           | scan tbl=242 name=Internal Worktable    | f            |            |
| 1      | 187435 | 3   | 6   | 1    | 531     | 308     | 387  | 0      |          |           | project                                 | f            |            |
| 1      | 187435 | 3   | 6   | 2    | 531     | 308     | 387  | 222912 |          |           | save tbl=245                            | f            | 38063308   |
| 1      | 187435 | 4   | 7   | 0    | 390     | 390     | 0    | 0      |          |           | scan tbl=238 name=Internal Worktable    | f            |            |
| 1      | 187435 | 4   | 7   | 1    | 390     | 390     | 0    | 0      |          |           | dist                                    | f            |            |
| 1      | 187435 | 4   | 8   | 0    | 1218    | 1066    | 0    | 0      |          |           | scan tbl=134954 name=Internal Worktable | f            |            |
| 1      | 187435 | 4   | 8   | 1    | 1218    | 1066    | 0    | 0      |          |           | project                                 | f            |            |
| 1      | 187435 | 4   | 8   | 2    | 1218    | 1066    | 0    | 0      |          |           | save tbl=245                            | f            | 37434163   |
| 1      | 187435 | 5   | 9   | 0    | 171     | 83      | 387  | 222912 |          |           | scan tbl=245 name=Internal Worktable    | f            |            |
| 1      | 187435 | 5   | 9   | 1    | 171     | 83      | 387  | 60120  |          |           | dist                                    | f            |            |
| 1      | 187435 | 5   | 10  | 0    | 3579    | 3383    | 387  | 222912 |          |           | scan tbl=134955 name=Internal Worktable | f            |            |
| 1      | 187435 | 5   | 10  | 1    | 3579    | 3383    | 387  | 0      |          |           | project                                 | f            |            |
| 1      | 187435 | 5   | 10  | 2    | 3579    | 3383    | 0    | 0      |          |           | hjoin tbl=256                           | f            |            |
| 1      | 187435 | 5   | 10  | 3    | 3579    | 3383    | 0    | 0      |          |           | project                                 | f            |            |
| 1      | 187435 | 5   | 10  | 4    | 3579    | 3383    | 0    | 0      |          |           | sort tbl=259                            | f            | 36805017 ▼ |
| 1      | 187435 | 6   | 11  | 0    | 10      | 7       | 0    | 0      |          |           | scan tbl=259 name=Internal Worktable    | f            |            |
| 1      | 187435 | 6   | 11  | 1    | 10      | 7       | 0    | 0      |          |           | return                                  | f            |            |
| 1      | 187435 | 6   | 12  | 0    | 9       | 9       | 0    | 0      |          |           | merge                                   | f            |            |
| 1      | 187435 | 6   | 12  | 1    | 9       | 9       | 0    | 0      |          |           | project                                 | f            |            |
| 1      | 187435 | 6   | 12  | 2    | 9       | 9       | 0    | 0      |          |           | return                                  | f            |            |

If you are returning an unusually large volume of data, see [Very large result set \(p. 483\)](#) for recommended solutions.

8. See if the bytes value is high relative to the rows value for any step, in comparison to other steps. This pattern can indicate that you are selecting a lot of columns. For recommended solutions, see [Large SELECT list \(p. 483\)](#).

## Using the SVL\_QUERY\_REPORT view

To analyze query summary information by slice, do the following:

1. Run the following to determine your query ID:

```
select query, elapsed, substring
from svl_qlog
order by query
desc limit 5;
```

Examine the truncated query text in the substring field to determine which query value represents your query. If you have run the query more than once, use the query value from the row with the lower elapsed value. That is the row for the compiled version. If you have been running many queries, you can raise the value used by the LIMIT clause used to make sure that your query is included.

2. Select rows from SVL\_QUERY\_REPORT for your query. Order the results by segment, step, elapsed\_time, and rows:

```
select * from svl_query_report where query = MyQueryID order by segment, step,
elapsed_time, rows;
```

3. For each step, check to see that all slices are processing approximately the same number of rows:

| userid | query  | slice | segment | step | start_time          | end_time            | elapsed_time | rows | bytes  | label |
|--------|--------|-------|---------|------|---------------------|---------------------|--------------|------|--------|-------|
| 100    | 141696 | 4     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 2100         | 1000 | 31000  | scan  |
| 100    | 141696 | 5     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 420          | 1100 | 31700  | bcast |
| 100    | 141696 | 1     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 437          | 1099 | 31812  | bcast |
| 100    | 141696 | 3     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 490          | 1066 | 30108  | bcast |
| 100    | 141696 | 6     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 576          | 1108 | 32316  | bcast |
| 100    | 141696 | 4     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 583          | 1128 | 32484  | bcast |
| 100    | 141696 | 0     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 726          | 1079 | 30804  | bcast |
| 100    | 141696 | 7     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 2109         | 1150 | 33300  | bcast |
| 100    | 141696 | 2     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 2406         | 1068 | 31056  | bcast |
| 100    | 141696 | 2     | 1       | 0    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 3441         | 8798 | 253580 | scan  |

Also check to see that all slices are taking approximately the same amount of time:

| userid | query  | slice | segment | step | start_time          | end_time            | elapsed_time | rows | bytes  | label |
|--------|--------|-------|---------|------|---------------------|---------------------|--------------|------|--------|-------|
| 100    | 141696 | 4     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 2100         | 1000 | 31000  | scan  |
| 100    | 141696 | 5     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 420          | 1100 | 31700  | bcast |
| 100    | 141696 | 1     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 437          | 1099 | 31812  | bcast |
| 100    | 141696 | 3     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 490          | 1066 | 30108  | bcast |
| 100    | 141696 | 6     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 576          | 1108 | 32316  | bcast |
| 100    | 141696 | 4     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 583          | 1128 | 32484  | bcast |
| 100    | 141696 | 0     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 726          | 1079 | 30804  | bcast |
| 100    | 141696 | 7     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 2109         | 1150 | 33300  | bcast |
| 100    | 141696 | 2     | 0       | 2    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 2406         | 1068 | 31056  | bcast |
| 100    | 141696 | 2     | 1       | 0    | 2014-09-12 18:45:33 | 2014-09-12 18:45:33 | 3441         | 8798 | 253580 | scan  |

Large discrepancies in these values can indicate data distribution skew due to a suboptimal distribution style for this particular query. For recommended solutions, see [Suboptimal data distribution \(p. 482\)](#).

## Mapping the query plan to the query summary

It helps to map the operations from the query plan to the steps (identified by the label field values) in the query summary to get further details:

| Query plan operation         | Label field value | Description                                                                                                |
|------------------------------|-------------------|------------------------------------------------------------------------------------------------------------|
| Aggregate                    | AGGR              | Evaluates aggregate functions and GROUP BY conditions.                                                     |
| HashAggregate                |                   |                                                                                                            |
| GroupAggregate               |                   |                                                                                                            |
| DS_BCAST_INNER               | BCAST (broadcast) | Broadcasts an entire table or some set of rows (such as a filtered set of rows from a table) to all nodes. |
| Doesn't appear in query plan | DELETE            | Deletes rows from tables.                                                                                  |
| DS_DIST_NONE                 | DIST (distribute) | Distributes rows to nodes for parallel joining purposes or other parallel processing.                      |
| DS_DIST_ALL_NONE             |                   |                                                                                                            |
| DS_DIST_INNER                |                   |                                                                                                            |

| Query plan operation         | Label field value   | Description                                                                                                                                         |
|------------------------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| DS_DIST_ALL_INNER            |                     |                                                                                                                                                     |
| DS_DIST_ALL_BOTH             |                     |                                                                                                                                                     |
| HASH                         | HASH                | Builds hash table for use in hash joins.                                                                                                            |
| Hash Join                    | HJOIN (hash join)   | Performs a hash join of two tables or intermediate result sets.                                                                                     |
| Doesn't appear in query plan | INSERT              | Inserts rows into tables.                                                                                                                           |
| Limit                        | LIMIT               | Applies a LIMIT clause to result sets.                                                                                                              |
| Merge                        | MERGE               | Merges rows derived from parallel sort or join operations.                                                                                          |
| Merge Join                   | MJOIN (merge join)  | Performs a merge join of two tables or intermediate result sets.                                                                                    |
| Nested Loop                  | NLOOP (nested loop) | Performs a nested loop join of two tables or intermediate result sets.                                                                              |
| Doesn't appear in query plan | PARSE               | Parses strings into binary values for loading.                                                                                                      |
| Project                      | PROJECT             | Evaluates expressions.                                                                                                                              |
| Network                      | RETURN              | Returns rows to the leader or the client.                                                                                                           |
| Doesn't appear in query plan | SAVE                | Materializes rows for use in the next processing step.                                                                                              |
| Seq Scan                     | SCAN                | Scans tables or intermediate result sets.                                                                                                           |
| Sort                         | SORT                | Sorts rows or intermediate result sets as required by other subsequent operations (such as joins or aggregations) or to satisfy an ORDER BY clause. |
| Unique                       | UNIQUE              | Applies a SELECT DISTINCT clause or removes duplicates as required by other operations.                                                             |
| Window                       | WINDOW              | Computes aggregate and ranking window functions.                                                                                                    |

# Improving query performance

Following are some common issues that affect query performance, with instructions on ways to diagnose and resolve them.

## Topics

- [Table statistics missing or out of date \(p. 481\)](#)
- [Nested loop \(p. 481\)](#)
- [Hash join \(p. 481\)](#)
- [Ghost rows or uncommitted rows \(p. 482\)](#)
- [Unsorted or missorted rows \(p. 482\)](#)
- [Suboptimal data distribution \(p. 482\)](#)
- [Insufficient memory allocated to the query \(p. 483\)](#)
- [Suboptimal WHERE clause \(p. 483\)](#)
- [Insufficiently restrictive predicate \(p. 483\)](#)
- [Very large result set \(p. 483\)](#)
- [Large SELECT list \(p. 483\)](#)

## Table statistics missing or out of date

If table statistics are missing or out of date, you might see the following:

- A warning message in EXPLAIN command results.
- A missing statistics alert event in STL\_ALERT\_EVENT\_LOG. For more information, see [Reviewing query alerts \(p. 474\)](#).

To fix this issue, run [ANALYZE \(p. 663\)](#).

## Nested loop

If a nested loop is present, you might see a nested loop alert event in STL\_ALERT\_EVENT\_LOG. You can also identify this type of event by running the query at [Identifying queries with nested loops \(p. 485\)](#). For more information, see [Reviewing query alerts \(p. 474\)](#).

To fix this, review your query for cross-joins and remove them if possible. Cross-joins are joins without a join condition that result in the Cartesian product of two tables. They are typically run as nested loop joins, which are the slowest of the possible join types.

## Hash join

If a hash join is present, you might see the following:

- Hash and hash join operations in the query plan. For more information, see [Analyzing the query plan \(p. 475\)](#).
- An HJOIN step in the segment with the highest maxtime value in SVL\_QUERY\_SUMMARY. For more information, see [Using the SVL\\_QUERY\\_SUMMARY view \(p. 476\)](#).

To fix this issue, you can take a couple of approaches:

- Rewrite the query to use a merge join if possible. You can do this by specifying join columns that are both distribution keys and sort keys.
- If the HJOIN step in SVL\_QUERY\_SUMMARY has a very high value in the rows field compared to the rows value in the final RETURN step in the query, check whether you can rewrite the query to join on a unique column. When a query does not join on a unique column, such as a primary key, that increases the number of rows involved in the join.

## Ghost rows or uncommitted rows

If ghost rows or uncommitted rows are present, you might see an alert event in STL\_ALERT\_EVENT\_LOG that indicates excessive ghost rows. For more information, see [Reviewing query alerts \(p. 474\)](#).

To fix this issue, you can take a couple of approaches:

- Check the **Loads** tab of your Amazon Redshift console for active load operations on any of the query tables. If you see active load operations, wait for those to complete before taking action.
- If there are no active load operations, run [VACUUM \(p. 1015\)](#) on the query tables to remove deleted rows.

## Unsorted or missorted rows

If unsorted or missorted rows are present, you might see a very selective filter alert event in STL\_ALERT\_EVENT\_LOG. For more information, see [Reviewing query alerts \(p. 474\)](#).

You can also check to see if any of the tables in your query have large unsorted areas by running the query in [Identifying tables with data skew or unsorted rows \(p. 484\)](#).

To fix this issue, you can take a couple of approaches:

- Run [VACUUM \(p. 1015\)](#) on the query tables to re-sort the rows.
- Review the sort keys on the query tables to see if any improvements can be made. Remember to weigh the performance of this query against the performance of other important queries and the system overall before making any changes. For more information, see [Working with sort keys \(p. 60\)](#).

## Suboptimal data distribution

If data distribution is suboptimal, you might see the following:

- A serial execution, large broadcast, or large distribution alert event appears in STL\_ALERT\_EVENT\_LOG. For more information, see [Reviewing query alerts \(p. 474\)](#).
- Slices are not processing approximately the same number of rows for a given step. For more information, see [Using the SVL\\_QUERY\\_REPORT view \(p. 478\)](#).
- Slices are not taking approximately the same amount of time for a given step. For more information, see [Using the SVL\\_QUERY\\_REPORT view \(p. 478\)](#).

If none of the preceding is true, you can also see if any of the tables in your query have data skew by running the query in [Identifying tables with data skew or unsorted rows \(p. 484\)](#).

To fix this issue, review the distribution styles for the tables in the query and see if any improvements can be made. Remember to weigh the performance of this query against the performance of other important queries and the system overall before making any changes. For more information, see [Working with data distribution styles \(p. 49\)](#).

## Insufficient memory allocated to the query

If insufficient memory is allocated to your query, you might see a step in SVL\_QUERY\_SUMMARY that has an `is_diskbased` value of true. For more information, see [Using the SVL\\_QUERY\\_SUMMARY view \(p. 476\)](#).

To fix this issue, allocate more memory to the query by temporarily increasing the number of query slots it uses. Workload Management (WLM) reserves slots in a query queue equivalent to the concurrency level set for the queue. For example, a queue with a concurrency level of 5 has 5 slots. Memory assigned to the queue is allocated equally to each slot. Assigning several slots to one query gives that query access to the memory for all of those slots. For more information on how to temporarily increase the slots for a query, see [wlm\\_query\\_slot\\_count \(p. 1702\)](#).

## Suboptimal WHERE clause

If your WHERE clause causes excessive table scans, you might see a SCAN step in the segment with the highest `maxtime` value in SVL\_QUERY\_SUMMARY. For more information, see [Using the SVL\\_QUERY\\_SUMMARY view \(p. 476\)](#).

To fix this issue, add a WHERE clause to the query based on the primary sort column of the largest table. This approach helps minimize scanning time. For more information, see [Amazon Redshift best practices for designing tables \(p. 15\)](#).

## Insufficiently restrictive predicate

If your query has an insufficiently restrictive predicate, you might see a SCAN step in the segment with the highest `maxtime` value in SVL\_QUERY\_SUMMARY that has a very high `rows` value compared to the `rows` value in the final RETURN step in the query. For more information, see [Using the SVL\\_QUERY\\_SUMMARY view \(p. 476\)](#).

To fix this issue, try adding a predicate to the query or making the existing predicate more restrictive to narrow the output.

## Very large result set

If your query returns a very large result set, consider rewriting the query to use [UNLOAD \(p. 991\)](#) to write the results to Amazon S3. This approach improves the performance of the RETURN step by taking advantage of parallel processing. For more information on checking for a very large result set, see [Using the SVL\\_QUERY\\_SUMMARY view \(p. 476\)](#).

## Large SELECT list

If your query has an unusually large SELECT list, you might see a `bytes` value that is high relative to the `rows` value for any step (in comparison to other steps) in SVL\_QUERY\_SUMMARY. This high `bytes` value can be an indicator that you are selecting a lot of columns. For more information, see [Using the SVL\\_QUERY\\_SUMMARY view \(p. 476\)](#).

To fix this issue, review the columns you are selecting and see if any can be removed.

## Diagnostic queries for query tuning

Use the following queries to identify issues with queries or underlying tables that can affect query performance. We recommend using these queries with the query tuning processes discussed in [Analyzing and improving queries \(p. 473\)](#).

### Topics

- [Identifying queries that are top candidates for tuning \(p. 484\)](#)
- [Identifying tables with data skew or unsorted rows \(p. 484\)](#)
- [Identifying queries with nested loops \(p. 485\)](#)
- [Reviewing queue wait times for queries \(p. 485\)](#)
- [Reviewing query alerts by table \(p. 486\)](#)
- [Identifying tables with missing statistics \(p. 486\)](#)

## Identifying queries that are top candidates for tuning

The following query identifies the top 50 most time-consuming statements that have been run in the last 7 days. You can use the results to identify queries that are taking unusually long. You can also identify queries that are run frequently (those that appear more than once in the result set). These queries are frequently good candidates for tuning to improve system performance.

This query also provides a count of the alert events associated with each query identified. These alerts provide details that you can use to improve the query's performance. For more information, see [Reviewing query alerts \(p. 474\)](#).

```
select trim(database) as db, count(query) as nqry,
max(substring (qrytext,1,80)) as qrytext,
min(run_minutes) as "min",
max(run_minutes) as "max",
avg(run_minutes) as "avg", sum(run_minutes) as total,
max(query) as max_query_id,
max(starttime)::date as last_run,
sum(alerts) as alerts, aborted
from (select userid, label, stl_query.query,
trim(database) as database,
trim(querytxt) as qrytext,
md5(trim(querytxt)) as qry_md5,
starttime, endtime,
(datediff(seconds, starttime,endtime)::numeric(12,2))/60 as run_minutes,
alrt.num_events as alerts, aborted
from stl_query
left outer join
(select query, 1 as num_events from stl_alert_event_log group by query) as alrt
on alrt.query = stl_query.query
where userid <> 1 and starttime >= dateadd(day, -7, current_date))
group by database, label, qry_md5, aborted
order by total desc limit 50;
```

## Identifying tables with data skew or unsorted rows

The following query identifies tables that have uneven data distribution (data skew) or a high percentage of unsorted rows.

A low skew value indicates that table data is properly distributed. If a table has a skew value of 4.00 or higher, consider modifying its data distribution style. For more information, see [Suboptimal data distribution \(p. 482\)](#).

If a table has a pct\_unsorted value greater than 20 percent, consider running the [VACUUM \(p. 1015\)](#) command. For more information, see [Unsorted or missorted rows \(p. 482\)](#).

Also review the mbytes and pct\_of\_total values for each table. These columns identify the size of the table and what percentage of raw disk space the table consumes. The raw disk space includes space that is reserved by Amazon Redshift for internal use, so it is larger than the nominal disk capacity, which is the amount of disk space available to the user. Use this information to verify that you have free disk space

equal to at least 2.5 times the size of your largest table. Having this space available enables the system to write intermediate results to disk when processing complex queries.

```

select trim(pgn.nspname) as schema,
trim(a.name) as table, id as tableid,
decode(pgc.reldiststyle,0, 'even',1,det.distkey ,8,'all') as distkey,
dist_ratio.ratio::decimal(10,4) as skew,
det.head_sort as "sortkey",
det.n_sortkeys as "#skns", b.mbytes,
decode(b.mbytes,0,0,((b.mbytes/part.total)::decimal)*100)::decimal(5,2)) as pct_of_total,
decode(det.max_enc,0,'n','y') as enc, a.rows,
decode(det.n_sortkeys, 0, null, a.unsorted_rows) as unsorted_rows ,
decode(det.n_sortkeys, 0, null, decode(a.rows,0,0, (a.unsorted_rows::decimal(32)/
a.rows)*100))::decimal(5,2) as pct_unsorted
from (select db_id, id, name, sum(rows) as rows,
sum(rows)-sum(sorted_rows) as unsorted_rows
from stv_tbl_perm a
group by db_id, id, name) as a
join pg_class as pgc on pgc.oid = a.id
join pg_namespace as pgn on pgn.oid = pgc.relnamespace
left outer join (select tbl, count(*) as mbytes
from stv_blocklist group by tbl) b on a.id=b.tbl
inner join (select attrelid,
min(case attisdistkey when 't' then attname else null end) as "distkey",
min(case attsortkeyord when 1 then attname else null end) as head_sort ,
max(attsortkeyord) as n_sortkeys,
max(attencodingtype) as max_enc
from pg_attribute group by 1) as det
on det.attrelid = a.id
inner join (select tbl, max(mbytes)::decimal(32)/min(mbytes) as ratio
from (select tbl, trim(name) as name, slice, count(*) as mbytes
from svv_diskusage group by tbl, name, slice)
group by tbl, name) as dist_ratio on a.id = dist_ratio.tbl
join (select sum(capacity) as total
from stv_partitions where part_begin=0) as part on 1=1
where mbytes is not null
order by mbytes desc;

```

## Identifying queries with nested loops

The following query identifies queries that have had alert events logged for nested loops. For information on how to fix the nested loop condition, see [Nested loop \(p. 481\)](#).

```

select query, trim(querytxt) as SQL, starttime
from stl_query
where query in (
select distinct query
from stl_alert_event_log
where event like 'Nested Loop Join in the query plan%')
order by starttime desc;

```

## Reviewing queue wait times for queries

The following query shows how long recent queries waited for an open slot in a query queue before running. If you see a trend of high wait times, you might want to modify your query queue configuration for better throughput. For more information, see [Implementing manual WLM \(p. 499\)](#).

```

select trim(database) as DB , w.query,
substring(q.querytxt, 1, 100) as querytxt, w.queue_start_time,
w.service_class as class, w.slot_count as slots,
w.total_queue_time/1000000 as queue_seconds,

```

```
w.total_exec_time/1000000 exec_seconds, (w.total_queue_time+w.total_Exec_time)/1000000 as
total_seconds
from stl_wlm_query w
left join stl_query q on q.query = w.query and q.userid = w.userid
where w.queue_start_Time >= dateadd(day, -7, current_Date)
and w.total_queue_Time > 0 and w.userid >1
and q.starttime >= dateadd(day, -7, current_Date)
order by w.total_queue_time desc, w.queue_start_time desc limit 35;
```

## Reviewing query alerts by table

The following query identifies tables that have had alert events logged for them, and also identifies what type of alerts are most frequently raised.

If the minutes value for a row with an identified table is high, check that table to see if it needs routine maintenance, such as having [ANALYZE \(p. 663\)](#) or [VACUUM \(p. 1015\)](#) run against it.

If the count value is high for a row but the table value is null, run a query against `STL_ALERT_EVENT_LOG` for the associated event value to investigate why that alert is getting raised so often.

```
select trim(s.perm_table_name) as table,
(sum(abs(datediff(seconds, s starttime, s endtime)))/60)::numeric(24,0) as minutes,
trim(split_part(l.event,':',1)) as event, trim(l.solution) as solution,
max(l.query) as sample_query, count(*)
from stl_alert_event_log as l
left join stl_scan as s on s.query = l.query and s.slice = l.slice
and s.segment = l.segment and s.step = l.step
where l.event_time >= dateadd(day, -7, current_Date)
group by 1,3,4
order by 2 desc,6 desc;
```

## Identifying tables with missing statistics

The following query provides a count of the queries that you are running against tables that are missing statistics. If this query returns any rows, look at the `plannode` value to determine the affected table, and then run [ANALYZE \(p. 663\)](#) on it.

```
select substring(trim(plannode),1,100) as plannode, count(*)
from stl_explain
where plannode like '%missing statistics%'
group by plannode
order by 2 desc;
```

# Troubleshooting queries

This section provides a quick reference for identifying and addressing some of the most common and most serious issues that you are likely to encounter with Amazon Redshift queries.

## Topics

- [Connection fails \(p. 487\)](#)
- [Query hangs \(p. 487\)](#)
- [Query takes too long \(p. 488\)](#)
- [Load fails \(p. 489\)](#)
- [Load takes too long \(p. 489\)](#)

- [Load data is incorrect \(p. 489\)](#)
- [Setting the JDBC fetch size parameter \(p. 489\)](#)

These suggestions give you a starting point for troubleshooting. You can also refer to the following resources for more detailed information.

- [Accessing Amazon Redshift clusters and databases](#)
- [Working with automatic table optimization \(p. 36\)](#)
- [Loading data \(p. 63\)](#)
- [Tutorial: Loading data from Amazon S3 \(p. 124\)](#)

## Connection fails

Your query connection can fail for the following reasons; we suggest the following troubleshooting approaches.

### **Client cannot connect to server**

If you are using SSL or server certificates, first remove this complexity while you troubleshoot the connection issue. Then add SSL or server certificates back when you have found a solution. For more information, go to [Configure Security Options for Connections](#) in the *Amazon Redshift Management Guide*.

### **Connection is refused**

Generally, when you receive an error message indicating that there is a failure to establish a connection, it means that there is an issue with the permission to access the cluster. For more information, go to [The connection is refused or fails](#) in the *Amazon Redshift Management Guide*.

## Query hangs

Your query can hang, or stop responding, for the following reasons; we suggest the following troubleshooting approaches.

### **Connection to the database is dropped**

Reduce the size of maximum transmission unit (MTU). The MTU size determines the maximum size, in bytes, of a packet that can be transferred in one Ethernet frame over your network connection. For more information, go to [The connection to the database is dropped](#) in the *Amazon Redshift Management Guide*.

### **Connection to the database times out**

Your client connection to the database appears to hang or time out when running long queries, such as a COPY command. In this case, you might observe that the Amazon Redshift console displays that the query has completed, but the client tool itself still appears to be running the query. The results of the query might be missing or incomplete depending on when the connection stopped. This effect happens when idle connections are terminated by an intermediate network component. For more information, go to [Firewall Timeout Issue](#) in the *Amazon Redshift Management Guide*.

### **Client-side out-of-memory error occurs with ODBC**

If your client application uses an ODBC connection and your query creates a result set that is too large to fit in memory, you can stream the result set to your client application by using a cursor. For more information, see [DECLARE \(p. 855\)](#) and [Performance considerations when using cursors \(p. 856\)](#).

#### **Client-side out-of-memory error occurs with JDBC**

When you attempt to retrieve large result sets over a JDBC connection, you might encounter client-side out-of-memory errors. For more information, see [Setting the JDBC fetch size parameter \(p. 489\)](#).

#### **There is a potential deadlock**

If there is a potential deadlock, try the following:

- View the [STV\\_LOCKS \(p. 1586\)](#) and [STL\\_TR\\_CONFLICT \(p. 1561\)](#) system tables to find conflicts involving updates to more than one table.
- Use the [PG\\_CANCEL\\_BACKEND \(p. 1383\)](#) function to cancel one or more conflicting queries.
- Use the [PG\\_TERMINATE\\_BACKEND \(p. 1384\)](#) function to terminate a session, which forces any currently running transactions in the terminated session to release all locks and roll back the transaction.
- Schedule concurrent write operations carefully. For more information, see [Managing concurrent write operations \(p. 118\)](#).

## Query takes too long

Your query can take too long for the following reasons; we suggest the following troubleshooting approaches.

#### **Tables are not optimized**

Set the sort key, distribution style, and compression encoding of the tables to take full advantage of parallel processing. For more information, see [Working with automatic table optimization \(p. 36\)](#)

#### **Query is writing to disk**

Your queries might be writing to disk for at least part of the query execution. For more information, see [Improving query performance \(p. 481\)](#).

#### **Query must wait for other queries to finish**

You might be able to improve overall system performance by creating query queues and assigning different types of queries to the appropriate queues. For more information, see [Implementing workload management \(p. 491\)](#).

#### **Queries are not optimized**

Analyze the explain plan to find opportunities for rewriting queries or optimizing the database. For more information, see [Query plan \(p. 465\)](#).

#### **Query needs more memory to run**

If a specific query needs more memory, you can increase the available memory by increasing the [wlm\\_query\\_slot\\_count \(p. 1702\)](#).

#### **Database requires a VACUUM command to be run**

Run the VACUUM command whenever you add, delete, or modify a large number of rows, unless you load your data in sort key order. The VACUUM command reorganizes your data to maintain the sort order and restore performance. For more information, see [Vacuuming tables \(p. 110\)](#).

## Load fails

Your data load can fail for the following reasons; we suggest the following troubleshooting approaches.

### Data Source is in a different AWS Region

By default, the Amazon S3 bucket or Amazon DynamoDB table specified in the COPY command must be in the same AWS Region as the cluster. If your data and your cluster are in different Regions, you receive an error similar to the following:

The bucket you are attempting to access must be addressed using the specified endpoint.

If at all possible, make sure your cluster and your data source are in the same Region. You can specify a different Region by using the [REGION \(p. 686\)](#) option with the COPY command.

#### Note

If your cluster and your data source are in different AWS Regions, you incur data transfer costs. You also have higher latency.

### COPY command fails

Query STL\_LOAD\_ERRORS to discover the errors that occurred during specific loads. For more information, see [STL\\_LOAD\\_ERRORS \(p. 1526\)](#).

## Load takes too long

Your load operation can take too long for the following reasons; we suggest the following troubleshooting approaches.

### COPY loads data from a single file

Split your load data into multiple files. When you load all the data from a single large file, Amazon Redshift is forced to perform a serialized load, which is much slower. The number of files should be a multiple of the number of slices in your cluster, and the files should be about equal size, between 1 MB and 1 GB after compression. For more information, see [Amazon Redshift best practices for designing queries \(p. 21\)](#).

### Load operation uses multiple COPY commands

If you use multiple concurrent COPY commands to load one table from multiple files, Amazon Redshift is forced to perform a serialized load, which is much slower. In this case, use a single COPY command.

## Load data is incorrect

Your COPY operation can load incorrect data in the following ways; we suggest the following troubleshooting approaches.

### Wrong files are loaded

Using an object prefix to specify data files can cause unwanted files to be read. Instead, use a manifest file to specify exactly which files to load. For more information, see the [copy\\_from\\_s3\\_manifest\\_file \(p. 685\)](#) option for the COPY command and [Example: COPY from Amazon S3 using a manifest \(p. 727\)](#) in the COPY examples.

## Setting the JDBC fetch size parameter

By default, the JDBC driver collects all the results for a query at one time. As a result, when you attempt to retrieve a large result set over a JDBC connection, you might encounter a client-side out-of-memory

error. To enable your client to retrieve result sets in batches instead of in a single all-or-nothing fetch, set the JDBC fetch size parameter in your client application.

**Note**

Fetch size is not supported for ODBC.

For the best performance, set the fetch size to the highest value that does not lead to out of memory errors. A lower fetch size value results in more server trips, which prolong execution times. The server reserves resources, including the WLM query slot and associated memory, until the client retrieves the entire result set or the query is canceled. When you tune the fetch size appropriately, those resources are released more quickly, making them available to other queries.

**Note**

If you need to extract large datasets, we recommend using an [UNLOAD \(p. 991\)](#) statement to transfer the data to Amazon S3. When you use UNLOAD, the compute nodes work in parallel to speed up the transfer of data.

For more information about setting the JDBC fetch size parameter, go to [Getting results based on a cursor](#) in the PostgreSQL documentation.

# Implementing workload management

You can use workload management (WLM) to define multiple query queues and to route queries to the appropriate queues at runtime.

In some cases, you might have multiple sessions or users running queries at the same time. In these cases, some queries might consume cluster resources for long periods of time and affect the performance of other queries. For example, suppose that one group of users submits occasional complex, long-running queries that select and sort rows from several large tables. Another group frequently submits short queries that select only a few rows from one or two tables and run in a few seconds. In this situation, the short-running queries might have to wait in a queue for a long-running query to complete. WLM helps manage this situation.

You can configure Amazon Redshift WLM to run with either automatic WLM or manual WLM.

## Automatic WLM

To maximize system throughput and use resources effectively, you can enable Amazon Redshift to manage how resources are divided to run concurrent queries with automatic WLM. **Automatic WLM** manages the resources required to run queries. Amazon Redshift determines how many queries run concurrently and how much memory is allocated to each dispatched query. You can enable automatic WLM using the Amazon Redshift console by choosing **Switch WLM mode** and then choosing **Auto WLM**. With this choice, up to eight queues are used to manage queries, and the **Memory** and **Concurrency on main** fields are both set to **Auto**. You can specify a priority that reflects the business priority of the workload or users that map to each queue. The default priority of queries is set to **Normal**. For information about how to change the priority of queries in a queue, see [Query priority \(p. 495\)](#). For more information, see [Implementing automatic WLM \(p. 493\)](#).

At runtime, you can route queries to these queues according to user groups or query groups. You can also configure a query monitoring rule (QMR) to limit long-running queries.

Working with concurrency scaling and automatic WLM, you can support virtually unlimited concurrent users and concurrent queries, with consistently fast query performance. For more information, see [Working with concurrency scaling \(p. 515\)](#).

### Note

We recommend that you create a parameter group and choose automatic WLM to manage your query resources. For details about how to migrate from manual WLM to automatic WLM, see [Migrating from manual WLM to automatic WLM \(p. 492\)](#).

## Manual WLM

Alternatively, you can manage system performance and your users' experience by modifying your WLM configuration to create separate queues for the long-running queries and the short-running queries. At runtime, you can route queries to these queues according to user groups or query groups. You can enable this manual configuration using the Amazon Redshift console by switching to **Manual WLM**. With this choice, you specify the queues used to manage queries, and the **Memory** and **Concurrency on main** field values. With a manual configuration, you can configure up to eight query queues and set the number of queries that can run in each of those queues concurrently.

You can set up rules to route queries to particular queues based on the user running the query or labels that you specify. You can also configure the amount of memory allocated to each queue, so that large queries run in queues with more memory than other queues. You can also configure a query monitoring rule (QMR) to limit long-running queries. For more information, see [Implementing manual WLM \(p. 499\)](#).

**Note**

We recommend configuring your manual WLM query queues with a total of 15 or fewer query slots. For more information, see [Concurrency level \(p. 500\)](#).

*WLM queuing limitations*

Note that in regards to a manual WLM configuration, the maximum slots you can allocate to a queue is 50. However, this doesn't mean that in an automatic WLM configuration, a Amazon Redshift cluster always runs 50 queries concurrently. This can change, based on the memory needs or other types of resource allocation on the cluster.

**Topics**

- [Modifying the WLM configuration \(p. 492\)](#)
- [Implementing automatic WLM \(p. 493\)](#)
- [Implementing manual WLM \(p. 499\)](#)
- [Working with concurrency scaling \(p. 515\)](#)
- [Working with short query acceleration \(p. 518\)](#)
- [WLM queue assignment rules \(p. 520\)](#)
- [Assigning queries to queues \(p. 523\)](#)
- [WLM dynamic and static configuration properties \(p. 524\)](#)
- [WLM query monitoring rules \(p. 527\)](#)
- [WLM system tables and views \(p. 533\)](#)

## Modifying the WLM configuration

The easiest way to modify the WLM configuration is by using the Amazon Redshift console. You can also use the AWS CLI or the Amazon Redshift API.

When you switch your cluster between automatic and manual WLM, your cluster is put into pending reboot state. The change doesn't take effect until the next cluster reboot.

For detailed information about modifying WLM configurations, see [Configuring Workload Management](#) in the *Amazon Redshift Management Guide*.

## Migrating from manual WLM to automatic WLM

To maximize system throughput and use resources most effectively, we recommend that you set up automatic WLM for your queues. Consider taking the following approach to set up a smooth transition from manual WLM to automatic WLM.

To migrate from manual WLM to automatic WLM and use query priorities, we recommend that you create a new parameter group, and then attach that parameter group to your cluster. For more information, see [Amazon Redshift Parameter Groups](#) in the *Amazon Redshift Management Guide*.

**Important**

To change the parameter group or to switch from manual to automatic WLM requires a cluster reboot. For more information, see [WLM dynamic and static configuration properties \(p. 524\)](#).

Let's take an example where there are three manual WLM queues. One each for an ETL workload, an analytics workload, and a data science workload. The ETL workload runs every 6 hours, the analytics workload runs throughout the day, and the data science workload can spike at any time. With manual WLM, you specify the memory and concurrency that each workload queue gets based on your understanding of the importance of each workload to the business. Specifying the memory and concurrency is not only hard to figure out, it also results in cluster resources being statically partitioned and thereby wasted when only a subset of the workloads is running.

You can use automatic WLM with query priorities to indicate the relative priorities of the workloads, avoiding the preceding issues. For this example, follow these steps:

- Create a new parameter group and switch to **Auto WLM** mode.
- Add queues for each of the three workloads: ETL workload, analytics workload, and data science workload. Use the same user groups for each workload that was used with **Manual WLM** mode.
- Set the priority for the ETL workload to High, the analytics workload to Normal, and the data science to Low. These priorities reflect your business priorities for the different workloads or user groups.
- Optionally, enable concurrency scaling for the analytics or data science queue so that queries in these queues get consistent performance even when the ETL workload is running every 6 hours.

With query priorities, when only the analytics workload is running on the cluster, it gets the entire system to itself. This yields high throughput with better system utilization. However, when the ETL workload starts, it gets the right of the way since it has a higher priority. Queries running as part of the ETL workload get priority during admission, in addition to preferential resource allocation after they are admitted. As a consequence, the ETL workload performs predictably regardless of what else might be running on the system. The predictable performance for a high priority workload comes at the cost of other, lower priority workloads that run longer either because their queries are waiting behind more important queries to complete. Or, because they are getting a smaller fraction of resources when they are running concurrently with higher priority queries. The scheduling algorithms used by Amazon Redshift facilitate that the lower priority queries do not suffer from starvation, but rather continue to make progress albeit at a slower pace.

#### Note

- The timeout field is not available in automatic WLM. Instead, use the QMR rule, `query_execution_time`. For more information, see [WLM query monitoring rules \(p. 527\)](#).
- The QMR action, HOP, is not applicable to automatic WLM. Instead, use the change priority action. For more information, see [WLM query monitoring rules \(p. 527\)](#).
- Within a parameter group, avoid mixing automatic WLM queues and manual WLM queues. Instead, create a new parameter group when migrating to automatic WLM.

## Implementing automatic WLM

With automatic workload management (WLM), Amazon Redshift manages query concurrency and memory allocation. You can create up to eight queues with the service class identifiers 100–107. Each queue has a priority. For more information, see [Query priority \(p. 495\)](#).

Automatic WLM determines the amount of resources that queries need and adjusts the concurrency based on the workload. When queries requiring large amounts of resources are in the system (for example, hash joins between large tables), the concurrency is lower. When lighter queries (such as inserts, deletes, scans, or simple aggregations) are submitted, concurrency is higher.

Automatic WLM is separate from short query acceleration (SQA) and it evaluates queries differently. Automatic WLM and SQA work together to allow short running and lightweight queries to complete even while long running, resource intensive queries are active. For more information about SQA, see [Working with short query acceleration \(p. 518\)](#).

Amazon Redshift enables automatic WLM through parameter groups:

- If your clusters use the default parameter group, Amazon Redshift enables automatic WLM for them.
- If your clusters use custom parameter groups, you can configure the clusters to enable automatic WLM. We recommend that you create a separate parameter group for your automatic WLM configuration.

To configure WLM, edit the `wlm_json_configuration` parameter in a parameter group that can be associated with one or more clusters. For more information, see [Modifying the WLM configuration \(p. 492\)](#).

You define query queues within the WLM configuration. You can add additional query queues to the default WLM configuration, up to a total of eight user queues. You can configure the following for each query queue:

- Priority
- Concurrency scaling mode
- User groups
- Query groups
- Query monitoring rules

## Priority

You can define the relative importance of queries in a workload by setting a priority value. The priority is specified for a queue and inherited by all queries associated with the queue. For more information, see [Query priority \(p. 495\)](#).

## Concurrency scaling mode

When concurrency scaling is enabled, Amazon Redshift automatically adds additional cluster capacity when you need it to process an increase in concurrent read and write queries. Your users see the most current data, whether the queries run on the main cluster or on a concurrency scaling cluster.

You manage which queries are sent to the concurrency scaling cluster by configuring WLM queues. When you enable concurrency scaling for a queue, eligible queries are sent to the concurrency scaling cluster instead of waiting in a queue. For more information, see [Working with concurrency scaling \(p. 515\)](#).

## User groups

You can assign a set of user groups to a queue by specifying each user group name or by using wildcards. When a member of a listed user group runs a query, that query runs in the corresponding queue. There is no set limit on the number of user groups that can be assigned to a queue. For more information, see [Assigning queries to queues based on user groups \(p. 523\)](#).

## Query groups

You can assign a set of query groups to a queue by specifying each query group name or by using wildcards. A *query group* is simply a label. At runtime, you can assign the query group label to a series of queries. Any queries that are assigned to a listed query group run in the corresponding queue. There is no set limit to the number of query groups that can be assigned to a queue. For more information, see [Assigning a query to a query group \(p. 523\)](#).

## Wildcards

If wildcards are enabled in the WLM queue configuration, you can assign user groups and query groups to a queue either individually or by using Unix shell-style wildcards. The pattern matching is case-insensitive.

For example, the '\*' wildcard character matches any number of characters. Thus, if you add dba\_\* to the list of user groups for a queue, any user-run query that belongs to a group with a name that begins with dba\_ is assigned to that queue. Examples are dba\_admin or DBA\_primary. The '?' wildcard character matches any single character. Thus, if the queue includes user-group dba?1, then user groups named dba11 and dba21 match, but dba12 doesn't match.

By default, wildcards aren't enabled.

## Query monitoring rules

Query monitoring rules define metrics-based performance boundaries for WLM queues and specify what action to take when a query goes beyond those boundaries. For example, for a queue dedicated to short running queries, you might create a rule that cancels queries that run for more than 60 seconds. To track poorly designed queries, you might have another rule that logs queries that contain nested loops. For more information, see [WLM query monitoring rules \(p. 527\)](#).

## Checking for automatic WLM

To check whether automatic WLM is enabled, run the following query. If the query returns at least one row, then automatic WLM is enabled.

```
select * from stv_wlm_service_class_config
where service_class >= 100;
```

The following query shows the number of queries that went through each query queue (service class). It also shows the average execution time, the number of queries with wait time at the 90th percentile, and the average wait time. Automatic WLM queries use service classes 100 through 107.

```
select final_state, service_class, count(*), avg(total_exec_time),
percentile_cont(0.9) within group (order by total_queue_time), avg(total_queue_time)
from stl_wlm_query where userid >= 100 group by 1,2 order by 2,1;
```

To find which queries were run by automatic WLM, and completed successfully, run the following query.

```
select a.queue_start_time, a.total_exec_time, label, trim(querytxt)
from stl_wlm_query a, stl_query b
where a.query = b.query and a.service_class >= 100 and a.final_state = 'Completed'
order by b.query desc limit 5;
```

## Query priority

Not all queries are of equal importance, and often performance of one workload or set of users might be more important. If you have enabled [automatic WLM \(p. 493\)](#), you can define the relative importance of queries in a workload by setting a priority value. The priority is specified for a queue and inherited by all queries associated with the queue. You associate queries to a queue by mapping user groups and query groups to the queue. You can set the following priorities (listed from highest to lowest priority):

1. HIGHEST

2. HIGH
3. NORMAL
4. LOW
5. LOWEST

Administrators use these priorities to show the relative importance of their workloads when there are queries with different priorities contending for the same resources. Amazon Redshift uses the priority when letting queries into the system, and to determine the amount of resources allocated to a query. By default, queries run with their priority set to NORMAL.

An additional priority, CRITICAL, which is a higher priority than HIGHEST, is available to superusers. To set this priority, you can use the functions [CHANGE\\_QUERY\\_PRIORITY \(p. 1379\)](#), [CHANGE\\_SESSION\\_PRIORITY \(p. 1380\)](#), and [CHANGE\\_USER\\_PRIORITY \(p. 1381\)](#). To grant a database user permission to use these functions, you can create a stored procedure and grant permission to a user. For an example, see [CHANGE\\_SESSION\\_PRIORITY \(p. 1380\)](#).

**Note**

Only one CRITICAL query can run at a time.

Let's take an example where the priority of an extract, transform, load (ETL) workload is higher than the priority of the analytics workload. The ETL workload runs every six hours, and the analytics workload runs throughout the day. When only the analytics workload is running on the cluster, it gets the entire system to itself, yielding high throughput with optimal system utilization. However, when the ETL workload starts, it gets the right of the way because it has a higher priority. Queries running as part of the ETL workload get the right of the way during admission and also preferential resource allocation after they are admitted. As a consequence, the ETL workload performs predictably regardless of what else might be running on the system. Thus, it provides predictable performance and the ability for administrators to provide service level agreements (SLAs) for their business users.

Within a given cluster, the predictable performance for a high priority workload comes at the cost of other, lower priority workloads. Lower priority workloads might run longer either because their queries are waiting behind more important queries to complete. Or they might run longer because they're getting a smaller fraction of resources when they are running concurrently with higher priority queries. Lower priority queries don't suffer from starvation, but rather keep making progress at a slower pace.

In the preceding example, the administrator can enable [concurrency scaling \(p. 515\)](#) for the analytics workload. Doing this enables that workload to maintain its throughput, even though the ETL workload is running at high priority.

## Configuring queue priority

If you have enabled automatic WLM, each queue has a priority value. Queries are routed to queues based on user groups and query groups. Start with a queue priority set to NORMAL. Set the priority higher or lower based on the workload associated with the queue's user groups and query groups.

You can change the priority of a queue on the Amazon Redshift console. On the Amazon Redshift console, the **Workload Management** page displays the queues and enables editing of queue properties such as **Priority**. To set the priority using the CLI or API operations, use the `wlm_json_configuration` parameter. For more information, see [Configuring Workload Management](#) in the *Amazon Redshift Management Guide*.

The following `wlm_json_configuration` example defines three user groups (`ingest`, `reporting`, and `analytics`). Queries submitted from users from one of these groups run with priority highest, normal, and low, respectively.

```
[
 {
```

```

 "user_group": [
 "ingest"
],
 "priority": "highest",
 "queue_type": "auto"
},
{
 "user_group": [
 "reporting"
],
 "priority": "normal",
 "queue_type": "auto"
},
{
 "user_group": [
 "analytics"
],
 "priority": "low",
 "queue_type": "auto",
 "auto_wlm": true
}
]

```

## Changing query priority with query monitoring rules

Query monitoring rules (QMR) enable you to change the priority of a query based on its behavior while it is running. You do this by specifying the priority attribute in a QMR predicate in addition to an action. For more information, see [WLM query monitoring rules \(p. 527\)](#).

For example, you can define a rule to cancel any query classified as high priority that runs for more than 10 minutes.

```

"rules": [
{
 "rule_name": "rule_abort",
 "predicate": [
 {
 "metric_name": "query_cpu_time",
 "operator": ">",
 "value": 600
 },
 {
 "metric_name": "query_priority",
 "operator": "=",
 "value": "high"
 }
],
 "action": "abort"
}
]

```

Another example is to define a rule to change the query priority to lowest for any query with current priority normal that spills more than 1 TB to disk.

```

"rules": [
{
 "rule_name": "rule_change_priority",
 "predicate": [
 {
 "metric_name": "query_temp_blocks_to_disk",
 "operator": ">",
 "value": 1000000
 }
],
 "action": "change_priority"
}
]

```

```

 },
 {
 "metric_name": "query_priority",
 "operator": "=",
 "value": "normal"
 }
],
 "action": "change_query_priority",
 "value": "lowest"
}
]

```

## Monitoring query priority

To display priority for waiting and running queries, view the `query_priority` column in the `stv_wlm_query_state` system table.

| query          | service_cl | wlm_start_time             | state         | queue_time |
|----------------|------------|----------------------------|---------------|------------|
| query_priority |            |                            |               |            |
| 2673299        | 102        | 2019-06-24 17:35:38.866356 | QueuedWaiting | 265116     |
| Highest        |            |                            |               |            |
| 2673236        | 101        | 2019-06-24 17:35:33.313854 | Running       | 0          |
| Highest        |            |                            |               |            |
| 2673265        | 102        | 2019-06-24 17:35:33.523332 | Running       | 0          |
| 2673284        | 102        | 2019-06-24 17:35:38.477366 | Running       | 0          |
| Highest        |            |                            |               |            |
| 2673288        | 102        | 2019-06-24 17:35:38.621819 | Running       | 0          |
| Highest        |            |                            |               |            |
| 2673310        | 103        | 2019-06-24 17:35:39.068513 | QueuedWaiting | 62970      |
| 2673303        | 102        | 2019-06-24 17:35:38.968921 | QueuedWaiting | 162560     |
| 2673306        | 104        | 2019-06-24 17:35:39.002733 | QueuedWaiting | 128691     |
|                |            |                            |               | Lowest     |

To list query priority for completed queries, see the `query_priority` column in the `stl_wlm_query` system table.

```
select query, service_class as svclass, service_class_start_time as starttime,
 query_priority
 from stl_wlm_query
 order by 3 desc limit 10;
```

| query   | svclass | starttime                  | query_priority |
|---------|---------|----------------------------|----------------|
| 2723254 | 100     | 2019-06-24 18:14:50.780094 | Normal         |
| 2723251 | 102     | 2019-06-24 18:14:50.749961 | Highest        |
| 2723246 | 102     | 2019-06-24 18:14:50.725275 | Highest        |
| 2723244 | 103     | 2019-06-24 18:14:50.719241 | High           |
| 2723243 | 101     | 2019-06-24 18:14:50.699325 | Low            |
| 2723242 | 102     | 2019-06-24 18:14:50.692573 | Highest        |
| 2723239 | 101     | 2019-06-24 18:14:50.668535 | Low            |
| 2723237 | 102     | 2019-06-24 18:14:50.661918 | Highest        |
| 2723236 | 102     | 2019-06-24 18:14:50.643636 | Highest        |

To optimize the throughput of your workload, Amazon Redshift might modify the priority of user submitted queries. Amazon Redshift uses advanced machine learning algorithms to determine when this optimization benefits your workload and automatically applies it when all the following conditions are met.

- Automatic WLM is enabled.
- Only one WLM queue is defined.
- You have not defined query monitoring rules (QMRs) which set query priority. Such rules include the QMR metric `query_priority` or the QMR action `change_query_priority`. For more information, see [WLM query monitoring rules \(p. 527\)](#).

## Implementing manual WLM

With manual WLM, you can manage system performance and your users' experience by modifying the WLM configuration to create separate queues for the long-running queries and short-running queries.

When users run queries in Amazon Redshift, the queries are routed to query queues. Each query queue contains a number of query slots. Each queue is allocated a portion of the cluster's available memory. A queue's memory is divided among the queue's query slots. You can enable Amazon Redshift to manage query concurrency with automatic WLM. For more information, see [Implementing automatic WLM \(p. 493\)](#).

Or you can configure WLM properties for each query queue. You do so to specify the way that memory is allocated among slots and how queries can be routed to specific queues at runtime. You can also configure WLM properties to cancel long-running queries.

By default, Amazon Redshift configures the following query queues:

- **One superuser queue**

The superuser queue is reserved for superusers only and it can't be configured. Use this queue only when you need to run queries that affect the system or for troubleshooting purposes. For example, use this queue when you need to cancel a user's long-running query or to add users to the database. Don't use it to perform routine queries. The queue doesn't appear in the console, but it does appear in the system tables in the database as the fifth queue. To run a query in the superuser queue, a user must be logged in as a superuser, and must run the query using the predefined superuser query group.

- **One default user queue**

The default queue is initially configured to run five queries concurrently. When you use manual WLM, you can change the concurrency, timeout, and memory allocation properties for the default queue, but you cannot specify user groups or query groups. The default queue must be the last queue in the WLM configuration. Any queries that are not routed to other queues run in the default queue.

Query queues are defined in the WLM configuration. The WLM configuration is an editable parameter (`wlm_json_configuration`) in a parameter group, which can be associated with one or more clusters. For more information, see [Configuring Workload Management](#) in the *Amazon Redshift Management Guide*.

You can add additional query queues to the default WLM configuration, up to a total of eight user queues. You can configure the following for each query queue:

- Concurrency scaling mode
- Concurrency level
- User groups
- Query groups
- WLM memory percent to use
- WLM timeout
- WLM query queue hopping
- Query monitoring rules

## Concurrency scaling mode

When concurrency scaling is enabled, Amazon Redshift automatically adds additional cluster capacity when you need it to process an increase in concurrent read and write queries. Users see the most current data, whether the queries run on the main cluster or on a concurrency scaling cluster.

You manage which queries are sent to the concurrency scaling cluster by configuring WLM queues. When you enable concurrency scaling for a queue, eligible queries are sent to the concurrency scaling cluster instead of waiting in a queue. For more information, see [Working with concurrency scaling \(p. 515\)](#).

## Concurrency level

Queries in a queue run concurrently until they reach the WLM query slot count, or *concurrency* level, defined for that queue. Subsequent queries then wait in the queue.

### Note

WLM concurrency level is different from the number of concurrent user connections that can be made to a cluster. For more information, see [Connecting to a Cluster](#) in the *Amazon Redshift Management Guide*.

In an automatic WLM configuration, which is recommended, the concurrency level is set to **Auto**. Amazon Redshift dynamically allocates memory to queries, which subsequently determines how many to run concurrently. This is based on the resources required for both running and queued queries. Auto WLM isn't configurable. For more information, see [Implementing automatic WLM \(p. 493\)](#).

In a manual WLM configuration, Amazon Redshift statically allocates a fixed amount of memory to each queue. The queue's memory is split evenly among the query slots. To illustrate, if a queue is allocated 20% of a cluster's memory and has 10 slots, each query is allocated 2% of the cluster's memory. The memory allocation remains fixed regardless of the number of queries running concurrently. Because of this fixed memory allocation, queries that run entirely in memory when the slot count is 5 might write intermediate results to disk if the slot count is increased to 20. In this instance each query's share of the queue's memory is reduced from 1/5th to 1/20th. The additional disk I/O could degrade performance.

The maximum slot count across all user-defined queues is 50. This limits the total slots for all queues, including the default queue. The only queue that isn't subject to the limit is the reserved superuser queue.

By default, manual WLM queues have a concurrency level of 5. Your workload might benefit from a higher concurrency level in certain cases, such as the following:

- If many small queries are forced to wait for long-running queries, create a separate queue with a higher slot count and assign the smaller queries to that queue. A queue with a higher concurrency level has less memory allocated to each query slot, but the smaller queries require less memory.

### Note

If you enable short-query acceleration (SQA), WLM automatically prioritizes short queries over longer-running queries, so you don't need a separate queue for short queries for most workflows. For more information, see [Working with short query acceleration \(p. 518\)](#).

- If you have multiple queries that each access data on a single slice, set up a separate WLM queue to run those queries concurrently. Amazon Redshift assigns concurrent queries to separate slices, which allows multiple queries to run in parallel on multiple slices. For example, if a query is a simple aggregate with a predicate on the distribution key, the data for the query is located on a single slice.

## A manual WLM example

This example is a simple, manual WLM scenario to show how slots and memory can be allocated. You implement manual WLM with three queues, which are the following:

- *data-ingestion queue* – This is set up for ingesting data. It's allocated 20% of the cluster's memory and it has 5 slots. Subsequently, 5 queries can run concurrently in the queue and each is allocated 4% of the memory.
- *data-scientist queue* – This is designed for memory-intensive queries. It's allocated 40% of the cluster's memory and it has 5 slots. Subsequently, 5 queries can run concurrently and each is allocated 8% of the memory.
- *default queue* – This is designed for the majority of the users in the organization. This includes sales and accounting groups that typically have short or medium running queries that aren't complicated. It's allocated 40% of the cluster's memory and it has 40 slots. 40 queries can run concurrently in this queue, with each query allocated 1% of the memory. This is the maximum number of slots that can be allocated for this queue because between all queues the limit is 50.

If you're running automatic WLM and your workload requires more than 15 queries to run in parallel, we recommend turning on concurrency scaling. This is because increasing the query slot count above 15 might create contention for system resources and limit the overall throughput of a single cluster. With concurrency scaling, you can run hundreds of queries in parallel, up to a configured number of concurrency scaling clusters. The number of concurrency scaling clusters is controlled by [max\\_concurrency\\_scaling\\_clusters \(p. 1693\)](#). For more information about concurrency scaling, see [Working with concurrency scaling \(p. 515\)](#).

For more information, see [Improving query performance \(p. 481\)](#).

## User groups

You can assign a set of user groups to a queue by specifying each user group name or by using wildcards. When a member of a listed user group runs a query, that query runs in the corresponding queue. There is no set limit on the number of user groups that can be assigned to a queue. For more information, see [Assigning queries to queues based on user groups \(p. 523\)](#).

## Query groups

You can assign a set of query groups to a queue by specifying each query group name or by using wildcards. A query group is simply a label. At runtime, you can assign the query group label to a series of queries. Any queries that are assigned to a listed query group run in the corresponding queue. There is no set limit to the number of query groups that can be assigned to a queue. For more information, see [Assigning a query to a query group \(p. 523\)](#).

## Wildcards

If wildcards are enabled in the WLM queue configuration, you can assign user groups and query groups to a queue either individually or by using Unix shell-style wildcards. The pattern matching is case-insensitive.

For example, the '\*' wildcard character matches any number of characters. Thus, if you add dba\_\* to the list of user groups for a queue, any user-run query that belongs to a group with a name that begins with dba\_ is assigned to that queue. Examples are dba\_admin or DBA\_primary,. The '?' wildcard character matches any single character. Thus, if the queue includes user-group dba?1, then user groups named dba11 and dba21 match, but dba12 doesn't match.

Wildcards are turned off by default.

## WLM memory percent to use

In an automatic WLM configuration, memory percent is set to **auto**. For more information, see [Implementing automatic WLM \(p. 493\)](#).

In a manual WLM configuration, to specify the amount of available memory that is allocated to a query, you can set the `WLM Memory Percent to Use` parameter. By default, each user-defined queue is allocated an equal portion of the memory that is available for user-defined queries. For example, if you have four user-defined queues, each queue is allocated 25 percent of the available memory. The superuser queue has its own allocated memory and cannot be modified. To change the allocation, you assign an integer percentage of memory to each queue, up to a total of 100 percent. Any unallocated memory is managed by Amazon Redshift and can be temporarily given to a queue if the queue requests additional memory for processing.

For example, if you configure four queues, you can allocate memory as follows: 20 percent, 30 percent, 15 percent, 15 percent. The remaining 20 percent is unallocated and managed by the service.

## WLM timeout

`WLM timeout (max_execution_time)` is deprecated. Instead, create a query monitoring rule (QMR) using `query_execution_time` to limit the elapsed execution time for a query. For more information, see [WLM query monitoring rules \(p. 527\)](#).

To limit the amount of time that queries in a given WLM queue are permitted to use, you can set the `WLM timeout` value for each queue. The `timeout` parameter specifies the amount of time, in milliseconds, that Amazon Redshift waits for a query to run before either canceling or hopping the query. The timeout is based on query execution time and doesn't include time spent waiting in a queue.

WLM attempts to hop [CREATE TABLE AS \(p. 838\)](#) (CTAS) statements and read-only queries, such as `SELECT` statements. Queries that can't be hopped are canceled. For more information, see [WLM query queue hopping \(p. 502\)](#).

WLM timeout doesn't apply to a query that has reached the returning state. To view the state of a query, see the [STV\\_WLM\\_QUERY\\_STATE \(p. 1605\)](#) system table. `COPY` statements and maintenance operations, such as `ANALYZE` and `VACUUM`, are not subject to WLM timeout.

The function of WLM timeout is similar to the [statement\\_timeout \(p. 1699\)](#) configuration parameter. The difference is that, where the `statement_timeout` configuration parameter applies to the entire cluster, WLM timeout is specific to a single queue in the WLM configuration.

If [statement\\_timeout \(p. 1699\)](#) is also specified, the lower of `statement_timeout` and WLM timeout (`max_execution_time`) is used.

## Query monitoring rules

Query monitoring rules define metrics-based performance boundaries for WLM queues and specify what action to take when a query goes beyond those boundaries. For example, for a queue dedicated to short running queries, you might create a rule that cancels queries that run for more than 60 seconds. To track poorly designed queries, you might have another rule that logs queries that contain nested loops. For more information, see [WLM query monitoring rules \(p. 527\)](#).

## WLM query queue hopping

A query can be hopped due to a [WLM timeout \(p. 502\)](#) or a [query monitoring rule \(QMR\) hop action \(p. 527\)](#). You can only hop queries in a manual WLM configuration.

When a query is hopped, WLM attempts to route the query to the next matching queue based on the [WLM queue assignment rules \(p. 520\)](#). If the query doesn't match any other queue definition, the query is canceled. It's not assigned to the default queue.

## WLM timeout actions

The following table summarizes the behavior of different types of queries with a WLM timeout.

| Query type                             | Action              |
|----------------------------------------|---------------------|
| INSERT, UPDATE, and DELETE             | Cancel              |
| User-defined functions (UDFs)          | Cancel              |
| UNLOAD                                 | Cancel              |
| COPY                                   | Continue execution  |
| Maintenance operations                 | Continue execution  |
| Read-only queries in a returning state | Continue execution  |
| Read-only queries in a running state   | Reassign or restart |
| CREATE TABLE AS (CTAS), SELECT INTO    | Reassign or restart |

## WLM timeout queue hopping

WLM hops the following types of queries when they time out:

- Read-only queries, such as SELECT statements, that are in a WLM state of `running`. To find the WLM state of a query, view the `STATE` column on the [STV\\_WLM\\_QUERY\\_STATE \(p. 1605\)](#) system table.
- CREATE TABLE AS (CTAS) statements. WLM queue hopping supports both user-defined and system-generated CTAS statements.
- SELECT INTO statements.

Queries that aren't subject to WLM timeout continue running in the original queue until completion. The following types of queries aren't subject to WLM timeout:

- COPY statements
- Maintenance operations, such as ANALYZE and VACUUM
- Read-only queries, such as SELECT statements, that have reached a WLM state of `returning`. To find the WLM state of a query, view the `STATE` column on the [STV\\_WLM\\_QUERY\\_STATE \(p. 1605\)](#) system table.

Queries that aren't eligible for hopping by WLM timeout are canceled when they time out. The following types of queries are not eligible for hopping by a WLM timeout:

- INSERT, UPDATE, and DELETE statements
- UNLOAD statements
- User-defined functions (UDFs)

## WLM timeout reassigned and restarted queries

When a query is hopped and no matching queue is found, the query is canceled.

When a query is hopped and a matching queue is found, WLM attempts to reassign the query to the new queue. If a query can't be reassigned, it's restarted in the new queue, as described following.

A query is reassigned only if all of the following are true:

- A matching queue is found.
- The new queue has enough free slots to run the query. A query might require multiple slots if the [wlm\\_query\\_slot\\_count \(p. 1702\)](#) parameter was set to a value greater than 1.
- The new queue has at least as much memory available as the query currently uses.

If the query is reassigned, the query continues executing in the new queue. Intermediate results are preserved, so there is minimal effect on total execution time.

If the query can't be reassigned, the query is canceled and restarted in the new queue. Intermediate results are deleted. The query waits in the queue, then begins running when enough slots are available.

## QMR hop actions

The following table summarizes the behavior of different types of queries with a QMR hop action.

| Query type                             | Action                         |
|----------------------------------------|--------------------------------|
| COPY                                   | Continue execution             |
| Maintenance operations                 | Continue execution             |
| User-defined functions (UDFs)          | Continue execution             |
| UNLOAD                                 | Reassign or continue execution |
| INSERT, UPDATE, and DELETE             | Reassign or continue execution |
| Read-only queries in a returning state | Reassign or continue execution |
| Read-only queries in a running state   | Reassign or restart            |
| CREATE TABLE AS (CTAS), SELECT INTO    | Reassign or restart            |

To find whether a query that was hopped by QMR was reassigned, restarted, or canceled, query the [STL\\_WLM\\_RULE\\_ACTION \(p. 1574\)](#) system log table.

## QMR hop action reassigned and restarted queries

When a query is hopped and no matching queue is found, the query is canceled.

When a query is hopped and a matching queue is found, WLM attempts to reassign the query to the new queue. If a query can't be reassigned, it's restarted in the new queue or continues execution in the original queue, as described following.

A query is reassigned only if all of the following are true:

- A matching queue is found.
- The new queue has enough free slots to run the query. A query might require multiple slots if the [wlm\\_query\\_slot\\_count \(p. 1702\)](#) parameter was set to a value greater than 1.
- The new queue has at least as much memory available as the query currently uses.

If the query is reassigned, the query continues executing in the new queue. Intermediate results are preserved, so there is minimal effect on total execution time.

If a query can't be reassigned, the query is either restarted or continues execution in the original queue. If the query is restarted, the query is canceled and restarted in the new queue. Intermediate results are deleted. The query waits in the queue, then begins execution when enough slots are available.

# Tutorial: Configuring manual workload management (WLM) queues

## Overview

We recommend configuring automatic workload management (WLM) in Amazon Redshift. For more information about automatic WLM, see [Implementing workload management \(p. 491\)](#). However, if you need multiple WLM queues, this tutorial walks you through the process of configuring manual workload management (WLM) in Amazon Redshift. By configuring manual WLM, you can improve query performance and resource allocation in your cluster.

Amazon Redshift routes user queries to queues for processing. WLM defines how those queries are routed to the queues. By default, Amazon Redshift has two queues available for queries: one for superusers, and one for users. The superuser queue cannot be configured and can only process one query at a time. You should reserve this queue for troubleshooting purposes only. The user queue can process up to five queries at a time, but you can configure this by changing the concurrency level of the queue if needed.

When you have several users running queries against the database, you might find another configuration to be more efficient. For example, if some users run resource-intensive operations, such as VACUUM, these might have a negative impact on less-intensive queries, such as reports. You might consider adding additional queues and configuring them for different workloads.

**Estimated time:** 75 minutes

**Estimated cost:** 50 cents

## Prerequisites

You need an Amazon Redshift cluster, the sample TICKIT database, and the Amazon Redshift RSQL client tool. If you do not already have these set up, go to [Amazon Redshift Getting Started Guide](#) and [Amazon Redshift RSQL](#).

## Sections

- [Section 1: Understanding the default queue processing behavior \(p. 505\)](#)
- [Section 2: Modifying the WLM query queue configuration \(p. 509\)](#)
- [Section 3: Routing queries to queues based on user groups and query groups \(p. 510\)](#)
- [Section 4: Using wlm\\_query\\_slot\\_count to temporarily override the concurrency level in a queue \(p. 513\)](#)
- [Section 5: Cleaning up your resources \(p. 515\)](#)

## Section 1: Understanding the default queue processing behavior

Before you start to configure manual WLM, it's useful to understand the default behavior of queue processing in Amazon Redshift. In this section, you create two database views that return information from several system tables. Then you run some test queries to see how queries are routed by default. For more information about system tables, see [System tables and views reference \(p. 1409\)](#).

## Step 1: Create the WLM\_QUEUE\_STATE\_VW view

In this step, you create a view called WLM\_QUEUE\_STATE\_VW. This view returns information from the following system tables.

- [STV\\_WLM\\_CLASSIFICATION\\_CONFIG \(p. 1603\)](#)
- [STV\\_WLM\\_SERVICE\\_CLASS\\_CONFIG \(p. 1608\)](#)
- [STV\\_WLM\\_SERVICE\\_CLASS\\_STATE \(p. 1609\)](#)

You use this view throughout the tutorial to monitor what happens to queues after you change the WLM configuration. The following table describes the data that the WLM\_QUEUE\_STATE\_VW view returns.

| Column             | Description                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| queue              | The number associated with the row that represents a queue. Queue number determines the order of the queues in the database.         |
| description        | A value that describes whether the queue is available only to certain user groups, to certain query groups, or all types of queries. |
| slots              | The number of slots allocated to the queue.                                                                                          |
| mem                | The amount of memory, in MB per slot, allocated to the queue.                                                                        |
| max_execution_time | The amount of time a query is allowed to run before it is terminated.                                                                |
| user_*             | A value that indicates whether wildcard characters are allowed in the WLM configuration to match user groups.                        |
| query_*            | A value that indicates whether wildcard characters are allowed in the WLM configuration to match query groups.                       |
| queued             | The number of queries that are waiting in the queue to be processed.                                                                 |
| executing          | The number of queries that are currently running.                                                                                    |
| executed           | The number of queries that have been run.                                                                                            |

### To create the WLM\_QUEUE\_STATE\_VW view

1. Open [Amazon Redshift RSQL](#) and connect to your TICKIT sample database. If you do not have this database, see [Prerequisites \(p. 505\)](#).
2. Run the following query to create the WLM\_QUEUE\_STATE\_VW view.

```

create view WLM_QUEUE_STATE_VW as
select (config.service_class-5) as queue
, trim (class.condition) as description
, config.num_query_tasks as slots
, config.query_working_mem as mem
, config.max_execution_time as max_time
, config.user_group_wild_card as "user_*"
, config.query_group_wild_card as "query_*"
, state.num_queued_queries queued
, state.num_executing_queries executing
, state.num_executed_queries executed
from
STV_WLM_CLASSIFICATION_CONFIG class,

```

```
STV_WLM_SERVICE_CLASS_CONFIG config,
STV_WLM_SERVICE_CLASS_STATE state
where
class.action_service_class = config.service_class
and class.action_service_class = state.service_class
and config.service_class > 4
order by config.service_class;
```

3. Run the following query to see the information that the view contains.

```
select * from wlm_queue_state_vw;
```

The following is an example result.

| queue                                       | description | slots | mem | max_time | user_* | query_* | queued | executing | executed |
|---------------------------------------------|-------------|-------|-----|----------|--------|---------|--------|-----------|----------|
| 0 (super user) and (query group: superuser) |             | 1     | 357 | 0        | false  | false   | 0      | 0         | 0        |
| 1 (querytype: any)                          |             | 5     | 836 | 0        | false  | false   | 0      | 1         | 160      |

## Step 2: Create the WLM\_QUERY\_STATE\_VW view

In this step, you create a view called WLM\_QUERY\_STATE\_VW. This view returns information from the [STV\\_WLM\\_QUERY\\_STATE \(p. 1605\)](#) system table.

You use this view throughout the tutorial to monitor the queries that are running. The following table describes the data that the WLM\_QUERY\_STATE\_VW view returns.

| Column     | Description                                                       |
|------------|-------------------------------------------------------------------|
| query      | The query ID.                                                     |
| queue      | The queue number.                                                 |
| slot_count | The number of slots allocated to the query.                       |
| start_time | The time that the query started.                                  |
| state      | The state of the query, such as executing.                        |
| queue_time | The number of microseconds that the query has spent in the queue. |
| exec_time  | The number of microseconds that the query has been running.       |

### To create the WLM\_QUERY\_STATE\_VW view

1. In RSQL, run the following query to create the WLM\_QUERY\_STATE\_VW view.

```
create view WLM_QUERY_STATE_VW as
select query, (service_class-5) as queue, slot_count, trim(wlm_start_time) as
start_time, trim(state) as state, trim(queue_time) as queue_time, trim(exec_time) as
exec_time
from stv_wlm_query_state;
```

2. Run the following query to see the information that the view contains.

```
select * from wlm_query_state_vw;
```

The following is an example result.

| query | queue | slot_count | start_time          | state     | queue_time | exec_time |
|-------|-------|------------|---------------------|-----------|------------|-----------|
| 1249  | 1     | 1          | 2014-09-24 22:19:16 | Executing | 0          | 516       |

## Step 3: Run test queries

In this step, you run queries from multiple connections in RSQL and review the system tables to determine how the queries were routed for processing.

For this step, you need two RSQL windows open:

- In RSQL window 1, you run queries that monitor the state of the queues and queries using the views you already created in this tutorial.
- In RSQL window 2, you run long-running queries to change the results you find in RSQL window 1.

### To run the test queries

1. Open two RSQL windows. If you already have one window open, you only need to open a second window. You can use the same user account for both of these connections.
2. In RSQL window 1, run the following query.

```
select * from wlm_query_state_vw;
```

The following is an example result.

| query | queue | slot_count | start_time          | state     | queue_time | exec_time |
|-------|-------|------------|---------------------|-----------|------------|-----------|
| 1258  | 1     | 1          | 2014-09-24 22:21:03 | Executing | 0          | 549       |

This query returns a self-referential result. The query that is currently running is the SELECT statement from this view. A query on this view always returns at least one result. Compare this result with the result that occurs after starting the long-running query in the next step.

3. In RSQL window 2, run a query from the TICKIT sample database. This query should run for approximately a minute so that you have time to explore the results of the WLM\_QUEUE\_STATE\_VW view and the WLM\_QUERY\_STATE\_VW view that you created earlier. In some cases, you might find that the query doesn't run long enough for you to query both views. In these cases, you can increase the value of the filter on l.listid to make it run longer.

#### Note

To reduce query execution time and improve system performance, Amazon Redshift caches the results of certain types of queries in memory on the leader node. When result caching is enabled, subsequent queries run much faster. To prevent the query from running too quickly, disable result caching for the current session.

To turn off result caching for the current session, set the [enable\\_result\\_cache\\_for\\_session \(p. 1691\)](#) parameter to off, as shown following.

```
set enable_result_cache_for_session to off;
```

In RSQL window 2, run the following query.

```
select avg(l.priceperTicket*s.qtySold) from listing l, sales s where l.listid < 100000;
```

4. In RSQL window 1, query WLM\_QUEUE\_STATE\_VW and WLM\_QUERY\_STATE\_VW and compare the results to your earlier results.

```
select * from wlm_queue_state_vw;
```

```
select * from wlm_query_state_vw;
```

The following are example results.

| queue                                       | description | slots      | mem                 | max_time  | user_*     | query_*   | queued | executing | executed |
|---------------------------------------------|-------------|------------|---------------------|-----------|------------|-----------|--------|-----------|----------|
| 0 (super user) and (query group: superuser) |             | 1          | 357                 | 0         | false      | false     | 0      | 0         | 0        |
| 1 (querytype: any)                          |             | 5          | 836                 | 0         | false      | false     | 0      | 2         | 163      |
| query                                       | queue       | slot_count | start_time          | state     | queue_time | exec_time |        |           |          |
| 1267                                        | 1           | 1          | 2014-09-24 22:22:30 | Executing | 0          | 684       |        |           |          |
| 1265                                        | 1           | 1          | 2014-09-24 22:22:26 | Executing | 0          | 4080859   |        |           |          |

Note the following differences between your previous queries and the results in this step:

- There are two rows now in WLM\_QUERY\_STATE\_VW. One result is the self-referential query for running a SELECT operation on this view. The second result is the long-running query from the previous step.
- The executing column in WLM\_QUEUE\_STATE\_VW has increased from 1 to 2. This column entry means that there are two queries running in the queue.
- The executed column is incremented each time you run a query in the queue.

The WLM\_QUEUE\_STATE\_VW view is useful for getting an overall view of the queues and how many queries are being processed in each queue. The WLM\_QUERY\_STATE\_VW view is useful for getting a more detailed view of the individual queries that are currently running.

## Section 2: Modifying the WLM query queue configuration

Now that you understand how queues work by default, you can learn how to configure query queues using manual WLM. In this section, you create and configure a new parameter group for your cluster. You create two additional user queues and configure them to accept queries based on the queries' user group or query group labels. Any queries that don't get routed to one of these two queues are routed to the default queue at runtime.

### To create a manual WLM configuration in a parameter group

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Configurations**, then choose **Workload management** to display the **Workload management** page.
3. Choose **Create** to display the **Create parameter group** window.
4. Enter **WLMTutorial** for both **Parameter group name** and **Description**, and then choose **Create** to create the parameter group.

#### Note

The **Parameter group name** is converted to all lower case format when created.

5. On the **Workload management** page, choose the parameter group **wlmtutorial** to display the details page with tabs for **Parameters** and **Workload management**.
6. Confirm that you're on the **Workload management** tab, then choose **Switch WLM mode** to display the **Concurrency settings** window.
7. Choose **Manual WLM**, then choose **Save** to switch to manual WLM.
8. Choose **Edit workload queues**.
9. Choose **Add queue** twice to add two queues. Now there are three queues: **Queue 1**, **Queue 2**, and **Default queue**.
10. Enter information for each queue as follows:

- For **Queue 1**, enter **30** for **Memory (%)**, **2** for **Concurrency on main**, and **test** for **Query groups**. Leave the other settings with their default values.
- For **Queue 2**, enter **40** for **Memory (%)**, **3** for **Concurrency on main**, and **admin** for **User groups**. Leave the other settings with their default values.
- Don't make any changes to the **Default queue**. WLM assigns unallocated memory to the default queue.

11. Choose **Save** to save your settings.

Next, associate the parameter group that has the manual WLM configuration with a cluster.

#### To associate a parameter group with a manual WLM configuration with a cluster

1. Sign in to the AWS Management Console and open the Amazon Redshift console at <https://console.aws.amazon.com/redshift/>.
2. On the navigation menu, choose **Clusters**, then choose **Clusters** to display a list of your clusters.
3. Choose your cluster, such as `examplecluster` to display the details of the cluster. Then choose the **Properties** tab to display the properties of that cluster.
4. In the **Database configurations** section, choose **Edit, Edit parameter group** to display the parameter groups window.
5. For **Parameter groups** choose the `wlmtutorial` parameter group that you previously created.
6. Choose **Save changes** to associate the parameter group.

The cluster is modified with the changed parameter group. However, you need to reboot the cluster for the changes to also be applied to the database.

7. Choose your cluster, and then choose **Reboot** for **Actions**.

After the cluster is rebooted, its status returns to **Available**.

## Section 3: Routing queries to queues based on user groups and query groups

Now you have your cluster associated with a new parameter group and you've configured WLM. Next, run some queries to see how Amazon Redshift routes queries into queues for processing.

### Step 1: View query queue configuration in the database

First, verify that the database has the WLM configuration that you expect.

#### To view the query queue configuration

1. Open RSQL and run the following query. The query uses the `WLM_QUEUE_STATE_VW` view you created in [Step 1: Create the `WLM\_QUEUE\_STATE\_VW` view \(p. 506\)](#). If you already had a session connected to the database prior to the cluster reboot, you need to reconnect.

```
select * from wlm_queue_state_vw;
```

The following is an example result.

| queue                                       | description | slots | mem | max_time | user_* | query_* | queued | executing | executed |
|---------------------------------------------|-------------|-------|-----|----------|--------|---------|--------|-----------|----------|
| 0 (super user) and (query group: superuser) |             | 1     | 357 | 0        | false  | false   | 0      | 0         | 0        |
| 1 (query group: test)                       |             | 2     | 627 | 0        | false  | false   | 0      | 0         | 0        |
| 2 (user group: admin)                       |             | 3     | 557 | 0        | false  | false   | 0      | 0         | 0        |
| 3 (querytype: any)                          |             | 5     | 250 | 0        | false  | false   | 0      | 1         | 0        |

Compare these results to the results you received in [Step 1: Create the WLM\\_QUEUE\\_STATE\\_VW view \(p. 506\)](#). Notice that there are now two additional queues. Queue 1 is now the queue for the test query group, and queue 2 is the queue for the admin user group.

Queue 3 is now the default queue. The last queue in the list is always the default queue. That's the queue to which queries are routed by default if no user group or query group is specified in a query.

- Run the following query to confirm that your query now runs in queue 3.

```
select * from wlm_query_state_vw;
```

The following is an example result.

| query | queue | slot_count | start_time            | state     | queue_time | exec_time |
|-------|-------|------------|-----------------------|-----------|------------|-----------|
| 2144  | 3     |            | 1 2014-09-24 23:49:59 | Executing | 0          | 550430    |

## Step 2: Run a query using the query group queue

### To run a query using the query group queue

- Run the following query to route it to the test query group.

```
set query_group to test;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

- From the other RSQL window, run the following query.

```
select * from wlm_query_state_vw;
```

The following is an example result.

| query | queue | slot_count | start_time            | state     | queue_time | exec_time |
|-------|-------|------------|-----------------------|-----------|------------|-----------|
| 2168  | 1     |            | 1 2014-09-24 23:54:18 | Executing | 0          | 6343309   |
| 2170  | 3     |            | 1 2014-09-24 23:54:24 | Executing | 0          | 847       |

The query was routed to the test query group, which is queue 1 now.

- Select all from the queue state view.

```
select * from wlm_queue_state_vw;
```

You see a result similar to the following.

| queue                                       | description | slots | mem | max_time | user_* | query_* | queued | executing | executed |
|---------------------------------------------|-------------|-------|-----|----------|--------|---------|--------|-----------|----------|
| 0 (super user) and (query group: superuser) |             | 1     | 357 | 0        | false  | false   | 0      | 0         | 0        |
| 1(query group: test)                        |             | 2     | 627 | 0        | false  | false   | 0      | 1         | 0        |
| 2(user group: admin)                        |             | 3     | 557 | 0        | false  | false   | 0      | 0         | 0        |
| 3(querytype: any)                           |             | 5     | 250 | 0        | false  | false   | 0      | 1         | 3        |

- Now, reset the query group and run the long query again:

```
reset query_group;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

- Run the queries against the views to see the results.

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

The following are example results.

| queue                                       | description | slots      | mem                 | max_time  | user_*     | query_*   | queued | executing | executed |
|---------------------------------------------|-------------|------------|---------------------|-----------|------------|-----------|--------|-----------|----------|
| 0 (super user) and (query group: superuser) |             | 1          | 357                 | 0         | false      | false     | 0      | 0         | 0        |
| 1 (query group: test)                       |             | 2          | 627                 | 0         | false      | false     | 0      | 0         | 1        |
| 2 (user group: admin)                       |             | 3          | 557                 | 0         | false      | false     | 0      | 0         | 0        |
| 3 (querytype: any)                          |             | 5          | 250                 | 0         | false      | false     | 0      | 2         | 5        |
| query                                       | queue       | slot_count | start_time          | state     | queue_time | exec_time |        |           |          |
| 2186                                        | 3           | 1          | 2014-09-24 23:57:52 | Executing | 0          | 649       |        |           |          |
| 2184                                        | 3           | 1          | 2014-09-24 23:57:48 | Executing | 0          | 4137349   |        |           |          |

The result should be that the query is now running in queue 3 again.

### Step 3: Create a database user and group

Before you can run any queries in this queue, you need to create the user group in the database and add a user to the group. Then you log in with RSQL using the new user's credentials and run queries. You need to run queries as a superuser, such as the admin user, to create database users.

#### To create a new database user and user group

1. In the database, create a new database user named adminwlm by running the following command in an RSQL window.

```
create user adminwlm createuser password '123Admin';
```

2. Then, run the following commands to create the new user group and add your new adminwlm user to it.

```
create group admin;
alter group admin add user adminwlm;
```

### Step 4: Run a query using the user group queue

Next you run a query and route it to the user group queue. You do this when you want to route your query to a queue that is configured to handle the type of query you want to run.

#### To run a query using the user group queue

1. In RSQL window 2, run the following queries to switch to the adminwlm account and run a query as that user.

```
set session authorization 'adminwlm';
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

2. In RSQL window 1, run the following query to see the query queue that the queries are routed to.

```
select * from wlm_query_state_vw;
select * from wlm_queue_state_vw;
```

The following are example results.

| queue                                       | description | slots      | mem                 | max_time  | user_*     | query_*   | queued | executing | executed |
|---------------------------------------------|-------------|------------|---------------------|-----------|------------|-----------|--------|-----------|----------|
| 0 (super user) and (query group: superuser) |             | 1          | 357                 | 0         | false      | false     | 0      | 0         | 0        |
| 1 (query group: test)                       |             | 2          | 627                 | 0         | false      | false     | 0      | 0         | 1        |
| 2 (user group: admin)                       |             | 3          | 557                 | 0         | false      | false     | 0      | 1         | 0        |
| 3 (querytype: any)                          |             | 5          | 250                 | 0         | false      | false     | 0      | 1         | 8        |
| query                                       | queue       | slot_count | start_time          | state     | queue_time | exec_time |        |           |          |
| 2202                                        | 2           | 1          | 2014-09-25 00:01:38 | Executing | 0          | 4885796   |        |           |          |
| 2204                                        | 3           | 1          | 2014-09-25 00:01:43 | Executing | 0          | 650       |        |           |          |

The queue that this query ran in is queue 2, the admin user queue. Anytime you run queries logged in as this user, they run in queue 2 unless you specify a different query group to use. The chosen queue depends on the queue assignment rules. For more information, see [WLM queue assignment rules \(p. 520\)](#).

3. Now run the following query from RSQL window 2.

```
set query_group to test;
select avg(l.priceperTicket*s.qtySold) from listing l, sales s where l.listid <40000;
```

4. In RSQL window 1, run the following query to see the query queue that the queries are routed to.

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

The following are example results.

| queue                                       | description | slots | mem | max_time | user_* | query_* | queued | executing | executed |
|---------------------------------------------|-------------|-------|-----|----------|--------|---------|--------|-----------|----------|
| 0 (super user) and (query group: superuser) |             | 1     | 357 | 0        | false  | false   | 0      | 0         | 0        |
| 1(query group: test)                        |             | 2     | 627 | 0        | false  | false   | 0      | 1         | 1        |
| 2(user group: admin)                        |             | 3     | 557 | 0        | false  | false   | 0      | 0         | 1        |
| 3(querytype: any)                           |             | 5     | 250 | 0        | false  | false   | 0      | 1         | 10       |

| query | queue | slot_count | start_time          | state     | queue_time | exec_time |
|-------|-------|------------|---------------------|-----------|------------|-----------|
| 2218  | 1     | 1          | 2014-09-25 00:04:30 | Executing | 0          | 4819666   |
| 2220  | 3     | 1          | 2014-09-25 00:04:35 | Executing | 0          | 685       |

5. When you're done, reset the query group.

```
reset query_group;
```

## Section 4: Using wlm\_query\_slot\_count to temporarily override the concurrency level in a queue

Sometimes, users might temporarily need more resources for a particular query. If so, they can use the `wlm_query_slot_count` configuration setting to temporarily override the way slots are allocated in a query queue. *Slots* are units of memory and CPU that are used to process queries. You might override the slot count when you have occasional queries that take a lot of resources in the cluster, such as when you perform a VACUUM operation in the database.

You might find that users often need to set `wlm_query_slot_count` for certain types of queries. If so, consider adjusting the WLM configuration and giving users a queue that better suits the needs of their queries. For more information about temporarily overriding the concurrency level by using slot count, see [wlm\\_query\\_slot\\_count \(p. 1702\)](#).

### Step 1: Override the concurrency level using `wlm_query_slot_count`

For the purposes of this tutorial, we run the same long-running SELECT query. We run it as the `adminwlm` user using `wlm_query_slot_count` to increase the number of slots available for the query.

#### To override the concurrency level using `wlm_query_slot_count`

1. Increase the limit on the query to make sure that you have enough time to query the `WLM_QUERY_STATE_VW` view and see a result.

```
set wlm_query_slot_count to 3;
select avg(l.priceperTicket*s.qtySold) from listing l, sales s where l.listid <40000;
```

- Now, query WLM\_QUERY\_STATE\_VW with the admin user to see how the query is running.

```
select * from wlm_query_state_vw;
```

The following is an example result.

| query | queue | slot_count | start_time          | state     | queue_time | exec_time |
|-------|-------|------------|---------------------|-----------|------------|-----------|
| 2240  | 2     | 3          | 2014-09-25 00:08:45 | Executing | 0          | 3731414   |
| 2242  | 3     | 1          | 2014-09-25 00:08:49 | Executing | 0          | 596       |

Notice that the slot count for the query is 3. This count means that the query is using all three slots to process the query, allocating all of the resources in the queue to that query.

- Now, run the following query.

```
select * from WLM_QUEUE_STATE_VW;
```

The following is an example result.

| queue                                       | description | slots | mem | max_time | user_* | query_* | queued | executing | executed |
|---------------------------------------------|-------------|-------|-----|----------|--------|---------|--------|-----------|----------|
| 0 (super user) and (query group: superuser) |             | 1     | 357 | 0        | false  | false   | 0      | 0         | 0        |
| 1(query group: test)                        |             | 2     | 627 | 0        | false  | false   | 0      | 0         | 4        |
| 2(user group: admin)                        |             | 3     | 557 | 0        | false  | false   | 0      | 1         | 3        |
| 3(querytype: any)                           |             | 5     | 250 | 0        | false  | false   | 0      | 1         | 25       |

The wlm\_query\_slot\_count configuration setting is valid for the current session only. If that session expires, or another user runs a query, the WLM configuration is used.

- Reset the slot count and rerun the test.

```
reset wlm_query_slot_count;
select avg(l.priceperTicket*s.qtySold) from listing l, sales s where l.listid <40000;
```

The following are example results.

| queue                                       | description | slots | mem | max_time | user_* | query_* | queued | executing | executed |
|---------------------------------------------|-------------|-------|-----|----------|--------|---------|--------|-----------|----------|
| 0 (super user) and (query group: superuser) |             | 1     | 357 | 0        | false  | false   | 0      | 0         | 0        |
| 1(query group: test)                        |             | 2     | 627 | 0        | false  | false   | 0      | 0         | 2        |
| 2(user group: admin)                        |             | 3     | 557 | 0        | false  | false   | 0      | 1         | 2        |
| 3(querytype: any)                           |             | 5     | 250 | 0        | false  | false   | 0      | 1         | 14       |

| query | queue | slot_count | start_time          | state     | queue_time | exec_time |
|-------|-------|------------|---------------------|-----------|------------|-----------|
| 2260  | 2     | 1          | 2014-09-25 00:12:11 | Executing | 0          | 4042618   |
| 2262  | 3     | 1          | 2014-09-25 00:12:15 | Executing | 0          | 680       |

## Step 2: Run queries from different sessions

Next, run queries from different sessions.

### To run queries from different sessions

- In RSQL window 1 and 2, run the following to use the test query group.

```
set query_group to test;
```

- In RSQL window 1, run the following long-running query.

```
select avg(l.priceperTicket*s.qtySold) from listing l, sales s where l.listid <40000;
```

- As the long-running query is still going in RSQL window 1, run the following. These commands increase the slot count to use all the slots for the queue and then start running the long-running query.

```
set wlm_query_slot_count to 2;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

4. Open a third RSQL window and query the views to see the results.

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

The following are example results.

| queue                                       | description | slots | mem | max_time | user_*  | query_* | queued | executing | executed |
|---------------------------------------------|-------------|-------|-----|----------|---------|---------|--------|-----------|----------|
| 0 (super user) and (query group: superuser) |             | 1     | 357 |          | 0 false | false   | 0      | 0         | 0        |
| 1 (query group: test)                       |             | 2     | 627 |          | 0 false | false   | 1      | 1         | 2        |
| 2 (user group: admin)                       |             | 3     | 557 |          | 0 false | false   | 0      | 0         | 3        |
| 3 (querytype: any)                          |             | 5     | 250 |          | 0 false | false   | 0      | 1         | 18       |

| query | queue | slot_count | start_time          | state         | queue_time | exec_time |  |  |  |
|-------|-------|------------|---------------------|---------------|------------|-----------|--|--|--|
| 2286  | 1     | 2          | 2014-09-25 00:16:48 | QueuedWaiting | 3758950    | 0         |  |  |  |
| 2282  | 1     | 1          | 2014-09-25 00:16:33 | Executing     | 0          | 19335850  |  |  |  |
| 2288  | 3     | 1          | 2014-09-25 00:16:52 | Executing     | 0          | 666       |  |  |  |

Notice that the first query is using one of the slots allocated to queue 1 to run the query. In addition, notice that there is one query that is waiting in the queue (where queued is 1 and state is QueuedWaiting). After the first query completes, the second one begins running. This execution happens because both queries are routed to the test query group, and the second query must wait for enough slots to begin processing.

## Section 5: Cleaning up your resources

Your cluster continues to accrue charges as long as it is running. When you have completed this tutorial, return your environment to the previous state by following the steps in [Find Additional Resources and Reset Your Environment](#) in *Amazon Redshift Getting Started Guide*.

For more information about WLM, see [Implementing workload management \(p. 491\)](#).

# Working with concurrency scaling

With the Concurrency Scaling feature, you can support thousands of concurrent users and concurrent queries, with consistently fast query performance.

When you turn on concurrency scaling, Amazon Redshift automatically adds additional cluster capacity to process an increase in both read and write queries. Users see the most current data, whether the queries run on the main cluster or a concurrency-scaling cluster. You're charged for concurrency-scaling clusters only for the time they're actively running queries. For more information about pricing, see [Amazon Redshift pricing](#).

You manage which queries are sent to the concurrency-scaling cluster by configuring WLM queues. When you turn on concurrency scaling, eligible queries are sent to the concurrency-scaling cluster instead of waiting in a queue.

## Concurrency scaling capabilities

When you turn on concurrency scaling for a WLM queue, it works for read operations, such as dashboard queries. It also works for commonly used write operations, such as statements for data ingestion and processing.

## Concurrency scaling capabilities for write operations

Concurrency scaling supports frequently used write operations, such as extract, transform, and load (ETL) statements. Concurrency scaling for write operations is especially useful when you want to maintain consistent response times when your cluster receives a large number of requests. It improves throughput for write operations contending for resources on the main cluster.

Concurrency scaling supports COPY, INSERT, DELETE, and UPDATE statements. In some cases, you might follow DDL statements, such as CREATE, with write statements in the same commit block. In these cases, the write statements are not sent to the concurrency-scaling cluster.

When you accrue credit for concurrency scaling, this credit accrual applies to both read and write operations.

## Limitations for concurrency scaling

The following are limitations for using Amazon Redshift concurrency scaling:

- It doesn't support queries on tables that use interleaved sort keys.
- It doesn't support queries on temporary tables.
- It doesn't support queries that access external resources that are protected by restrictive network or virtual private cloud (VPC) configurations.
- It doesn't support queries that contain Python user-defined functions (UDFs) and  $\Lambda$  UDFs.
- It doesn't support queries that access system tables, PostgreSQL catalog tables, or no-backup tables.
- It doesn't support COPY or UNLOAD queries that access an external resource that has restrictive resource policies. External resources can include Amazon S3 buckets or DynamoDB tables. Policies can restrict access to a specific source VPC (aws:sourceVpc), source VPC endpoint (aws:sourceVpc), or source IP address (aws:SourceIp). In some cases, you might need to remove policies on an external resource. Doing this means that COPY or UNLOAD queries accessing the resource are sent to the concurrency-scaling cluster.
- Amazon Redshift concurrency scaling for write operations is not supported for DDL operations, such as CREATE TABLE or ALTER TABLE.
- It doesn't support ANALYZE for the COPY command.
- It doesn't support write operations on a target table where DISTSTYLE is set to ALL.
- It doesn't support COPY from Amazon Redshift Spectrum or Amazon EMR.
- It doesn't support write operations on tables with identity columns.
- Amazon Redshift supports concurrency scaling for write operations on only Amazon Redshift RA3 nodes, specifically ra3.16xlarge, ra3.4xlarge, and ra3.xlplus. Concurrency scaling for write operations isn't supported on other node types.
- Amazon Redshift concurrency scaling isn't supported for auto refresh for materialized views. For more information, see [Streaming ingestion](#).

## AWS Regions for concurrency scaling

Concurrency scaling is available in these AWS Regions:

- US East (N. Virginia) Region (us-east-1)
- US East (Ohio) Region (us-east-2)
- AWS GovCloud (US-East)
- US West (N. California) Region (us-west-1)
- US West (Oregon) Region (us-west-2)

- Asia Pacific (Mumbai) Region (ap-south-1)
- Asia Pacific (Seoul) Region (ap-northeast-2)
- Asia Pacific (Singapore) Region (ap-southeast-1)
- Asia Pacific (Sydney) Region (ap-southeast-2)
- Asia Pacific (Tokyo) Region (ap-northeast-1)
- Canada (Central) Region (ca-central-1)
- Europe (Frankfurt) Region (eu-central-1)
- Europe (Ireland) Region (eu-west-1)
- Europe (London) Region (eu-west-2)
- Europe (Paris) Region (eu-west-3)
- Europe (Stockholm) Region (eu-north-1)
- South America (São Paulo) Region (sa-east-1)

## Concurrency scaling candidates

Queries are routed to the concurrency scaling cluster only when the main cluster meets the following requirements:

- EC2-VPC platform.
- Node type must be dc2.8xlarge, ds2.8xlarge, dc2.large, ds2.xlarge, ra3.xlplus, ra3.4xlarge, or ra3.16xlarge. Concurrency scaling for write operations is supported on only the following Amazon Redshift RA3 nodes: ra3.16xlarge, ra3.4xlarge, and ra3.xlplus.
- Maximum of 32 compute nodes for clusters with ra3.xlplus, 8xlarge, or 16xlarge node types. In addition, the number of nodes of the main cluster can't be larger than 32 nodes when the cluster was originally created. For example, even if a cluster currently has 20 nodes, but was originally created with 40, it does not meet the requirements for concurrency scaling. Conversely, if a cluster currently has 40 nodes, but was originally created with 20, it does meet the requirements for concurrency scaling.
- Not a single-node cluster.

## Configuring concurrency scaling queues

You route queries to concurrency scaling clusters by enabling a workload manager (WLM) queue as a concurrency scaling queue. To turn on concurrency scaling for a queue, set the **Concurrency Scaling mode** value to **auto**.

When the number of queries routed to a concurrency scaling queue exceeds the queue's configured concurrency, eligible queries are sent to the concurrency scaling cluster. When slots become available, queries are run on the main cluster. The number of queues is limited only by the number of queues permitted per cluster. As with any WLM queue, you route queries to a concurrency scaling queue based on user groups or by labeling queries with query group labels. You can also route queries by defining [WLM query monitoring rules \(p. 527\)](#). For example, you might route all queries that take longer than 5 seconds to a concurrency scaling queue.

The default number of concurrency scaling clusters is one. The number of concurrency scaling clusters that can be used is controlled by [max\\_concurrency\\_scaling\\_clusters \(p. 1693\)](#).

## Monitoring concurrency scaling

You can see whether a query is running on the main cluster or a concurrency scaling cluster by navigating to **Cluster** in the Amazon Redshift console and choosing a cluster. Then choose the **Query monitoring** tab and **Workload concurrency** to view information about running queries and queued queries.

To find execution times, query the STL\_QUERY table and filter on the concurrency\_scaling\_status column. The following query compares the queue time and execution time for queries run on the concurrency scaling cluster and queries run on the main cluster.

```
SELECT w.service_class AS queue
 , q.concurrency_scaling_status
 , COUNT(*) AS queries
 , SUM(q.aborted) AS aborted
 , SUM(ROUND(total_queue_time::NUMERIC / 1000000,2)) AS queue_secs
 , SUM(ROUND(total_exec_time::NUMERIC / 1000000,2)) AS exec_secs
FROM stl_query q
JOIN stl_wlm_query w
 USING (userid,query)
WHERE q.userid > 1
 AND qstarttime > '2019-01-04 16:38:00'
 AND qendtime < '2019-01-04 17:40:00'
GROUP BY 1,2
ORDER BY 1,2;
```

## Concurrency scaling system views

A set of system views with the prefix SVCS provides details from the system log tables about queries on both the main and concurrency scaling clusters.

The following views have similar information as the corresponding STL views or SVL views:

- [SVCS\\_ALERT\\_EVENT\\_LOG \(p. 1611\)](#)
- [SVCS\\_COMPILE \(p. 1612\)](#)
- [SVCS\\_EXPLAIN \(p. 1614\)](#)
- [SVCS\\_PLAN\\_INFO \(p. 1616\)](#)
- [SVCS\\_QUERY\\_SUMMARY \(p. 1618\)](#)
- [SVCS\\_STREAM\\_SEGS \(p. 1626\)](#)

The following views are specific to concurrency scaling.

- [SVCS\\_CONCURRENCY\\_SCALING\\_USAGE \(p. 1614\)](#)

For more information about concurrency scaling, see the following topics in the *Amazon Redshift Management Guide*.

- [Viewing Concurrency Scaling Data](#)
- [Viewing Cluster Performance During Query Execution](#)
- [Viewing Query Details](#)

## Working with short query acceleration

Short query acceleration (SQA) prioritizes selected short-running queries ahead of longer-running queries. SQA runs short-running queries in a dedicated space, so that SQA queries aren't forced to wait in queues behind longer queries. SQA only prioritizes queries that are short-running and are in a user-defined queue. With SQA, short-running queries begin running more quickly and users see results sooner.

If you enable SQA, you can reduce workload management (WLM) queues that are dedicated to running short queries. In addition, long-running queries don't need to contend with short queries for slots in a queue, so you can configure your WLM queues to use fewer query slots. When you use lower concurrency, query throughput is increased and overall system performance is improved for most workloads.

[CREATE TABLE AS \(p. 838\)](#) (CTAS) statements and read-only queries, such as [SELECT \(p. 928\)](#) statements, are eligible for SQA.

Amazon Redshift uses a machine learning algorithm to analyze each eligible query and predict the query's execution time. By default, WLM dynamically assigns a value for the SQA maximum runtime based on analysis of your cluster's workload. Alternatively, you can specify a fixed value of 1–20 seconds. If the query's predicted run time is less than the defined or dynamically assigned SQA maximum runtime and the query is waiting in a WLM queue, SQA separates the query from the WLM queues and schedules it for priority execution. If a query runs longer than the SQA maximum runtime, WLM moves the query to the first matching WLM queue based on the [WLM queue assignment rules \(p. 520\)](#). Over time, predictions improve as SQA learns from your query patterns.

SQA is enabled by default in the default parameter group and for all new parameter groups. To disable SQA in the Amazon Redshift console, edit the WLM configuration for a parameter group and deselect **Enable short query acceleration**. As a best practice, we recommend using a WLM query slot count of 15 or fewer to maintain optimum overall system performance. For information about modifying WLM configurations, see [Configuring Workload Management](#) in the *Amazon Redshift Management Guide*.

## Maximum runtime for short queries

When you enable SQA, WLM sets the maximum runtime for short queries to dynamic by default. We recommend keeping the dynamic setting for SQA maximum runtime. You can override the default setting by specifying a fixed value of 1–20 seconds.

In some cases, you might consider using different values for the SQA maximum runtime values to improve your system performance. In such cases, analyze your workload to find the maximum execution time for most of your short-running queries. The following query returns the maximum runtime for queries at about the 70th percentile.

```
select least(greatest(percentile_cont(0.7)
within group (order by total_exec_time / 1000000) + 2, 2), 20)
from stl_wlm_query
where userid >= 100
and final_state = 'Completed';
```

After you identify a maximum runtime value that works well for your workload, you don't need to change it unless your workload changes significantly.

## Monitoring SQA

To check whether SQA is enabled, run the following query. If the query returns a row, then SQA is enabled.

```
select * from stv_wlm_service_class_config
where service_class = 14;
```

The following query shows the number of queries that went through each query queue (service class). It also shows the average execution time, the number of queries with wait time at the 90th percentile, and the average wait time. SQA queries use in service class 14.

```
select final_state, service_class, count(*), avg(total_exec_time),
percentile_cont(0.9) within group (order by total_queue_time), avg(total_queue_time)
from stl_wlm_query where userid >= 100 group by 1,2 order by 2,1;
```

To find which queries were picked up by SQA and completed successfully, run the following query.

```
select a.queue_start_time, a.total_exec_time, label, trim(querytxt)
from stl_wlm_query a, stl_query b
where a.query = b.query and a.service_class = 14 and a.final_state = 'Completed'
order by b.query desc limit 5;
```

To find queries that SQA picked up but that timed out, run the following query.

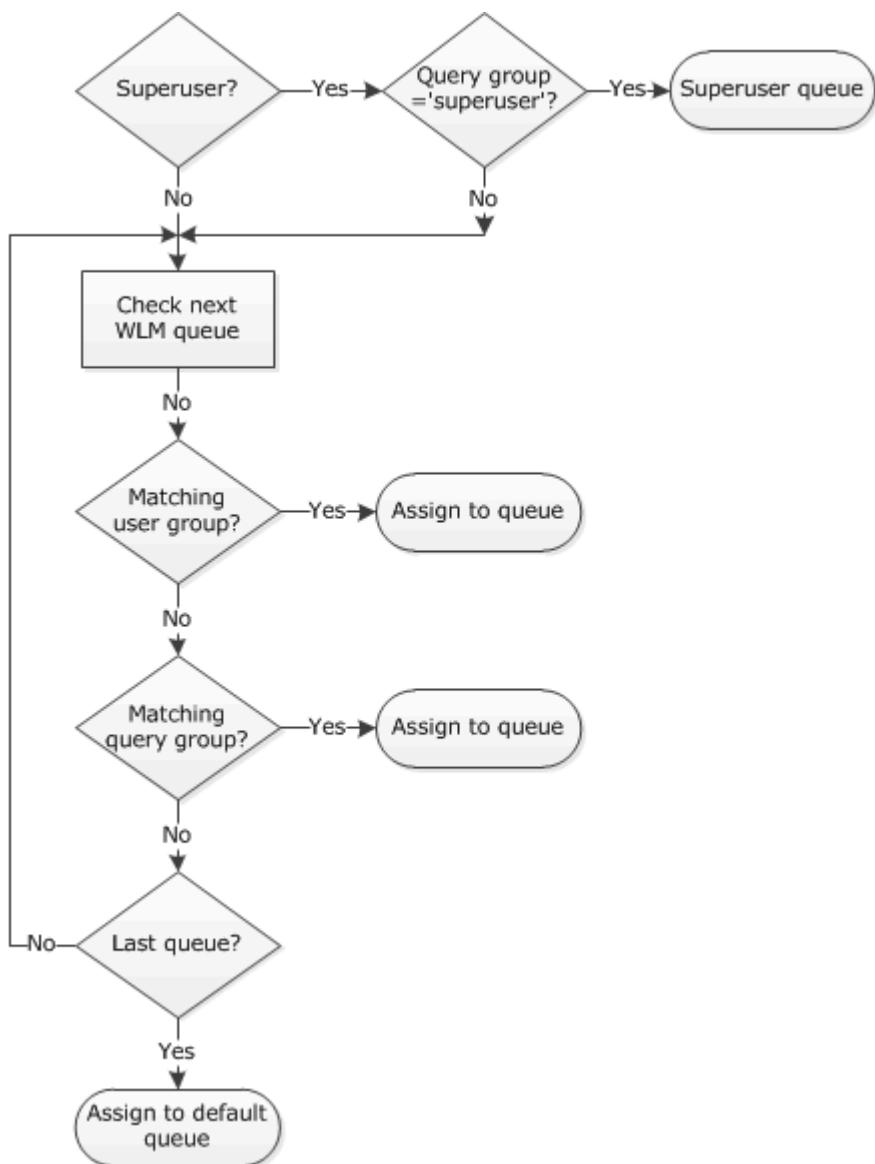
```
select a.queue_start_time, a.total_exec_time, label, trim(querytxt)
from stl_wlm_query a, stl_query b
where a.query = b.query and a.service_class = 14 and a.final_state = 'Evicted'
order by b.query desc limit 5;
```

## WLM queue assignment rules

When a user runs a query, WLM assigns the query to the first matching queue, based on the WLM queue assignment rules:

1. If a user is logged in as a superuser and runs a query in the query group labeled superuser, the query is assigned to the superuser queue.
2. If a user belongs to a listed user group or runs a query within a listed query group, the query is assigned to the first matching queue.
3. If a query doesn't meet any criteria, the query is assigned to the default queue, which is the last queue defined in the WLM configuration.

The following diagram illustrates how these rules work.

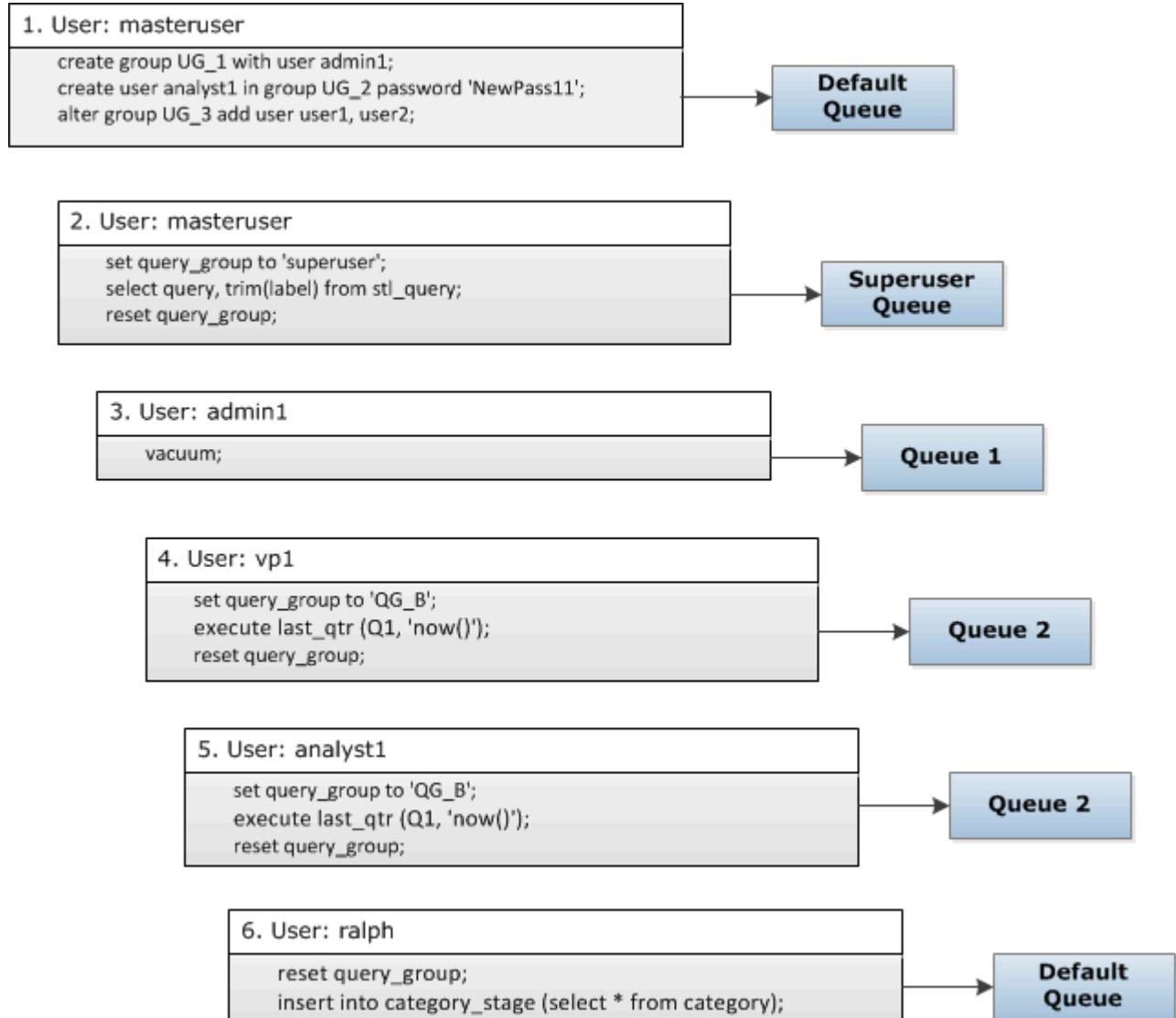


## Queue assignments example

The following table shows a WLM configuration with the superuser queue and four user-defined queues.

| Queue     | Concurrency | User Groups | Query Groups |
|-----------|-------------|-------------|--------------|
| Superuser | 1           |             | superuser    |
| 1         | 5           | UG_1        |              |
| 2         | 5           |             | QG_B         |
| 3         | 5           | UG_2        | QG_C         |
| Default   | 5           |             |              |

The following illustration shows how queries are assigned to the queues in the previous table according to user groups and query groups. For information about how to assign queries to user groups and query groups at runtime, see [Assigning queries to queues \(p. 523\)](#) later in this section.



In this example, WLM makes the following assignments:

1. The first set of statements shows three ways to assign users to user groups. The statements are run by the user `adminuser1`, which is not a member of a user group listed in any WLM queue. No query group is set, so the statements are routed to the default queue.
2. The user `adminuser1` is a superuser and the query group is set to '`superuser`', so the query is assigned to the superuser queue.
3. The user `admin1` is a member of the user group listed in queue 1, so the query is assigned to queue 1.
4. The user `vp1` is not a member of any listed user group. The query group is set to '`QG_B`', so the query is assigned to queue 2.
5. The user `analyst1` is a member of the user group listed in queue 3, but '`QG_B`' matches queue 2, so the query is assigned to queue 2.
6. The user `ralph` is not a member of any listed user group and the query group was reset, so there is no matching queue. The query is assigned to the default queue.

# Assigning queries to queues

The following examples assign queries to queues according to user groups and query groups.

## Assigning queries to queues based on user groups

If a user group name is listed in a queue definition, queries run by members of that user group are assigned to the corresponding queue. The following example creates user groups and adds users to groups by using the SQL commands [CREATE USER \(p. 846\)](#), [CREATE GROUP \(p. 792\)](#), and [ALTER GROUP \(p. 633\)](#).

```
create group admin_group with user admin246, admin135, sec555;
create user vp1234 in group ad_hoc_group password 'vpPass1234';
alter group admin_group add user analyst44, analyst45, analyst46;
```

## Assigning a query to a query group

You can assign a query to a queue at runtime by assigning your query to the appropriate query group. Use the `SET` command to begin a query group.

```
SET query_group TO group_label
```

Here, *group\_label* is a query group label that is listed in the WLM configuration.

All queries that you run after the `SET query_group` command run as members of the specified query group until you either reset the query group or end your current login session. For information about setting and resetting Amazon Redshift objects, see [SET \(p. 977\)](#) and [RESET \(p. 915\)](#) in the SQL Command Reference.

The query group labels that you specify must be included in the current WLM configuration; otherwise, the `SET query_group` command has no effect on query queues.

The label defined in the `TO` clause is captured in the query logs so that you can use the label for troubleshooting. For information about the `query_group` configuration parameter, see [query\\_group \(p. 1695\)](#) in the Configuration Reference.

The following example runs two queries as part of the query group 'priority' and then resets the query group.

```
set query_group to 'priority';
select count(*)from stv_blocklist;
select query, elapsed, substring from svt_qlog order by query desc limit 5;
reset query_group;
```

## Assigning queries to the superuser queue

To assign a query to the superuser queue, log on to Amazon Redshift as a superuser and then run the query in the superuser group. When you are done, reset the query group so that subsequent queries do not run in the superuser queue.

The following example assigns two commands to run in the superuser queue.

```
set query_group to 'superuser';
```

```
analyze;
vacuum;
reset query_group;
```

To view a list of superusers, query the PG\_USER system catalog table.

```
select * from pg_user where usesuper = 'true';
```

## WLM dynamic and static configuration properties

The WLM configuration properties are either dynamic or static. You can apply dynamic properties to the database without a cluster reboot, but static properties require a cluster reboot for changes to take effect. However, if you change dynamic and static properties at the same time, then you must reboot the cluster for all the property changes to take effect. This is true whether the changed properties are dynamic or static.

While dynamic properties are being applied, your cluster status is modifying. Switching between automatic WLM and manual WLM is a static change and requires a cluster reboot to take effect.

The following table indicates which WLM properties are dynamic or static when using automatic WLM or manual WLM.

| WLM Property                      | Automatic WLM  | Manual WLM     |
|-----------------------------------|----------------|----------------|
| Query groups                      | Dynamic        | Static         |
| Query group wildcard              | Dynamic        | Static         |
| User groups                       | Dynamic        | Static         |
| User group wildcard               | Dynamic        | Static         |
| Concurrency on main               | Not applicable | Dynamic        |
| Concurrency Scaling mode          | Dynamic        | Dynamic        |
| Enable short query acceleration   | Not applicable | Dynamic        |
| Maximum runtime for short queries | Dynamic        | Dynamic        |
| Percent of memory to use          | Not applicable | Dynamic        |
| Timeout                           | Not applicable | Dynamic        |
| Priority                          | Dynamic        | Not applicable |
| Adding or removing queues         | Dynamic        | Static         |

If you modify a query monitoring rule (QMR), the change happens automatically without the need to modify the cluster.

**Note**

When using manual WLM, if the timeout value is changed, the new value is applied to any query that begins running after the value is changed. If the concurrency or percent of memory to use

are changed, Amazon Redshift changes to the new configuration dynamically. Thus, currently running queries aren't affected by the change. For more information, see [WLM Dynamic Memory Allocation](#).

When using automatic WLM, timeout is ignored.

#### Topics

- [WLM dynamic memory allocation \(p. 525\)](#)
- [Dynamic WLM example \(p. 525\)](#)

## WLM dynamic memory allocation

In each queue, WLM creates a number of query slots equal to the queue's concurrency level. The amount of memory allocated to a query slot equals the percentage of memory allocated to the queue divided by the slot count. If you change the memory allocation or concurrency, Amazon Redshift dynamically manages the transition to the new WLM configuration. Thus, active queries can run to completion using the currently allocated amount of memory. At the same time, Amazon Redshift ensures that total memory usage never exceeds 100 percent of available memory.

The workload manager uses the following process to manage the transition:

1. WLM recalculates the memory allocation for each new query slot.
2. If a query slot is not actively being used by a running query, WLM removes the slot, which makes that memory available for new slots.
3. If a query slot is actively in use, WLM waits for the query to finish.
4. As active queries complete, the empty slots are removed and the associated memory is freed.
5. As enough memory becomes available to add one or more slots, new slots are added.
6. When all queries that were running at the time of the change finish, the slot count equals the new concurrency level, and the transition to the new WLM configuration is complete.

In effect, queries that are running when the change takes place continue to use the original memory allocation. Queries that are queued when the change takes place are routed to new slots as they become available.

If the WLM dynamic properties are changed during the transition process, WLM immediately begins to transition to the new configuration, starting from the current state. To view the status of the transition, query the [STV\\_WLM\\_SERVICE\\_CLASS\\_CONFIG \(p. 1608\)](#) system table.

## Dynamic WLM example

Suppose that your cluster WLM is configured with two queues, using the following dynamic properties.

| Queue | Concurrency | % Memory to Use |
|-------|-------------|-----------------|
| 1     | 4           | 50%             |
| 2     | 4           | 50%             |

Now suppose that your cluster has 200 GB of memory available for query processing. (This number is arbitrary and used for illustration only.) As the following equation shows, each slot is allocated 25 GB.

$$(200 \text{ GB} * 50\%) / 4 \text{ slots} = 25 \text{ GB}$$

Next, you change your WLM to use the following dynamic properties.

| Queue | Concurrency | % Memory to Use |
|-------|-------------|-----------------|
| 1     | 3           | 75%             |
| 2     | 4           | 25%             |

As the following equation shows, the new memory allocation for each slot in queue 1 is 50 GB.

$$(200 \text{ GB} * 75\%) / 3 \text{ slots} = 50 \text{ GB}$$

Suppose that queries A1, A2, A3, and A4 are running when the new configuration is applied, and queries B1, B2, B3, and B4 are queued. WLM dynamically reconfigures the query slots as follows.

| Step | Queries Running | Current Slot Count | Target Slot Count | Allocated Memory | Available Memory |
|------|-----------------|--------------------|-------------------|------------------|------------------|
| 1    | A1, A2, A3, A4  | 4                  | 0                 | 100 GB           | 50 GB            |
| 2    | A2, A3, A4      | 3                  | 0                 | 75 GB            | 75 GB            |
| 3    | A3, A4          | 2                  | 0                 | 50 GB            | 100 GB           |
| 4    | A3, A4, B1      | 2                  | 1                 | 100 GB           | 50 GB            |
| 5    | A4, B1          | 1                  | 1                 | 75 GB            | 75 GB            |
| 6    | A4, B1, B2      | 1                  | 2                 | 125 GB           | 25 GB            |
| 7    | B1, B2          | 0                  | 2                 | 100 GB           | 50 GB            |
| 8    | B1, B2, B3      | 0                  | 3                 | 150 GB           | 0 GB             |

1. WLM recalculates the memory allocation for each query slot. Originally, queue 1 was allocated 100 GB. The new queue has a total allocation of 150 GB, so the new queue immediately has 50 GB available. Queue 1 is now using four slots, and the new concurrency level is three slots, so no new slots are added.
2. When one query finishes, the slot is removed and 25 GB is freed. Queue 1 now has three slots and 75 GB of available memory. The new configuration needs 50 GB for each new slot, but the new concurrency level is three slots, so no new slots are added.
3. When a second query finishes, the slot is removed, and 25 GB is freed. Queue 1 now has two slots and 100 GB of free memory.
4. A new slot is added using 50 GB of the free memory. Queue 1 now has three slots, and 50 GB free memory. Queued queries can now be routed to the new slot.
5. When a third query finishes, the slot is removed, and 25 GB is freed. Queue 1 now has two slots, and 75 GB of free memory.
6. A new slot is added using 50 GB of the free memory. Queue 1 now has three slots, and 25 GB free memory. Queued queries can now be routed to the new slot.
7. When the fourth query finishes, the slot is removed, and 25 GB is freed. Queue 1 now has two slots and 50 GB of free memory.
8. A new slot is added using the 50 GB of free memory. Queue 1 now has three slots with 50 GB each and all available memory has been allocated.

The transition is complete and all query slots are available to queued queries.

## WLM query monitoring rules

In Amazon Redshift workload management (WLM), query monitoring rules define metrics-based performance boundaries for WLM queues and specify what action to take when a query goes beyond those boundaries. For example, for a queue dedicated to short running queries, you might create a rule that cancels queries that run for more than 60 seconds. To track poorly designed queries, you might have another rule that logs queries that contain nested loops.

You define query monitoring rules as part of your workload management (WLM) configuration. You can define up to 25 rules for each queue, with a limit of 25 rules for all queues. Each rule includes up to three conditions, or predicates, and one action. A *predicate* consists of a metric, a comparison condition (=, <, or >), and a value. If all of the predicates for any rule are met, that rule's action is triggered. Possible rule actions are log, hop, and abort, as discussed following.

The rules in a given queue apply only to queries running in that queue. A rule is independent of other rules.

WLM evaluates metrics every 10 seconds. If more than one rule is triggered during the same period, WLM initiates the most severe action—abort, then hop, then log. If the action is hop or abort, the action is logged and the query is evicted from the queue. If the action is log, the query continues to run in the queue. WLM initiates only one log action per query per rule. If the queue contains other rules, those rules remain in effect. If the action is hop and the query is routed to another queue, the rules for the new queue apply.

When all of a rule's predicates are met, WLM writes a row to the [STL\\_WLM\\_RULE\\_ACTION \(p. 1574\)](#) system table. In addition, Amazon Redshift records query metrics for currently running queries to [STV\\_QUERY\\_METRICS \(p. 1592\)](#). Metrics for completed queries are stored in [STL\\_QUERY\\_METRICS \(p. 1542\)](#).

## Defining a query monitoring rule

You create query monitoring rules as part of your WLM configuration, which you define as part of your cluster's parameter group definition.

You can create rules using the AWS Management Console or programmatically using JSON.

### Note

If you choose to create rules programmatically, we strongly recommend using the console to generate the JSON that you include in the parameter group definition. For more information, see [Creating or modifying a query monitoring rule using the console](#) and [Configuring Parameter Values Using the AWS CLI](#) in the *Amazon Redshift Management Guide*.

To define a query monitoring rule, you specify the following elements:

- A rule name – Rule names must be unique within the WLM configuration. Rule names can be up to 32 alphanumeric characters or underscores, and can't contain spaces or quotation marks. You can have up to 25 rules per queue, and the total limit for all queues is 25 rules.
- One or more predicates – You can have up to three predicates per rule. If all the predicates for any rule are met, the associated action is triggered. A predicate is defined by a metric name, an operator (=, <, or >), and a value. An example is `query_cpu_time > 100000`. For a list of metrics and examples of values for different metrics, see [Query monitoring metrics for Amazon Redshift \(p. 528\)](#) following in this section.

- An action – If more than one rule is triggered, WLM chooses the rule with the most severe action. Possible actions, in ascending order of severity, are:
  - Log – Record information about the query in the STL\_WLM\_RULE\_ACTION system table. Use the Log action when you want to only write a log record. WLM creates at most one log per query, per rule. Following a log action, other rules remain in force and WLM continues to monitor the query.
  - Hop (only available with manual WLM) – Log the action and hop the query to the next matching queue. If there isn't another matching queue, the query is canceled. QMR hops only [CREATE TABLE AS](#) (CTAS) statements and read-only queries, such as SELECT statements. For more information, see [WLM query queue hopping \(p. 502\)](#).
  - Abort – Log the action and cancel the query. QMR doesn't stop COPY statements and maintenance operations, such as ANALYZE and VACUUM.
  - Change priority (only available with automatic WLM) – Change the priority of a query.

To limit the runtime of queries, we recommend creating a query monitoring rule instead of using WLM timeout. For example, you can set max\_execution\_time to 50,000 milliseconds as shown in the following JSON snippet.

```
"max_execution_time": 50000
```

But we recommend instead that you define an equivalent query monitoring rule that sets query\_execution\_time to 50 seconds as shown in the following JSON snippet.

```
"rules": [
 {
 "rule_name": "rule_query_execution",
 "predicate": [
 {
 "metric_name": "query_execution_time",
 "operator": ">",
 "value": 50
 }
],
 "action": "abort"
 }
]
```

For steps to create or modify a query monitoring rule, see [Creating or modifying a query monitoring rule using the console](#) and [Properties in the wlm\\_json\\_configuration Parameter](#) in the *Amazon Redshift Management Guide*.

You can find more information about query monitoring rules in the following topics:

- [Query monitoring metrics for Amazon Redshift \(p. 528\)](#)
- [Query monitoring rules templates \(p. 532\)](#)
- [Creating a Rule Using the Console](#)
- [Configuring Workload Management](#)
- [System tables and views for query monitoring rules \(p. 532\)](#)

## Query monitoring metrics for Amazon Redshift

The following table describes the metrics used in query monitoring rules. (These metrics are distinct from the metrics stored in the [STV\\_QUERY\\_METRICS \(p. 1592\)](#) and [STL\\_QUERY\\_METRICS \(p. 1542\)](#) system tables.)

For a given metric, the performance threshold is tracked either at the query level or the segment level. For more information about segments and steps, see [Query planning and execution workflow \(p. 463\)](#).

**Note**

The [WLM timeout \(p. 502\)](#) parameter is distinct from query monitoring rules.

| Metric                     | Name                       | Description                                                                                                                                                                                                                                |
|----------------------------|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Query CPU time             | query_cpu_time             | CPU time used by the query, in seconds. CPU time is distinct from Query execution time.<br><br>Valid values are 0–999,999.                                                                                                                 |
| Blocks read                | query_blocks_read          | Number of 1 MB data blocks read by the query.<br><br>Valid values are 0–1,048,575.                                                                                                                                                         |
| Scan row count             | scan_row_count             | The number of rows in a scan step. The row count is the total number of rows emitted before filtering rows marked for deletion (ghost rows) and before applying user-defined query filters.<br><br>Valid values are 0–999,999,999,999,999. |
| Query execution time       | query_execution_time       | Elapsed execution time for a query, in seconds. Execution time doesn't include time spent waiting in a queue.<br><br>Valid values are 0–86,399.                                                                                            |
| Query queue time           | query_queue_time           | Time spent waiting in a queue, in seconds.<br><br>Valid values are 0–86,399.                                                                                                                                                               |
| CPU usage                  | query_cpu_usage_percent    | Percent of CPU capacity used by the query.<br><br>Valid values are 0–6,399.                                                                                                                                                                |
| Memory to disk             | query_temp_blocks_to_disk  | Temporary disk space used to write intermediate results, in 1 MB blocks.<br><br>Valid values are 0–319,815,679.                                                                                                                            |
| CPU skew                   | cpu_skew                   | The ratio of maximum CPU usage for any slice to average CPU usage for all slices. This metric is defined at the segment level.<br><br>Valid values are 0–99.                                                                               |
| I/O skew                   | io_skew                    | The ratio of maximum blocks read (I/O) for any slice to average blocks read for all slices. This metric is defined at the segment level.<br><br>Valid values are 0–99.                                                                     |
| Rows joined                | join_row_count             | The number of rows processed in a join step.<br><br>Valid values are 0–999,999,999,999,999.                                                                                                                                                |
| Nested loop join row count | nested_loop_join_row_count | The number of rows in a nested loop join.                                                                                                                                                                                                  |

| Metric                  | Name                    | Description                                                                                                                                                                                                                                    |
|-------------------------|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                         |                         | Valid values are 0–999,999,999,999,999.                                                                                                                                                                                                        |
| Return row count        | return_row_count        | The number of rows returned by the query.<br>Valid values are 0–999,999,999,999,999.                                                                                                                                                           |
| Segment execution time  | segment_execution_time  | Histogram execution time for a single segment, in seconds. To avoid or reduce sampling errors, include segment_execution_time > 10 in your rules.<br>Valid values are 0–86,388.                                                                |
| Spectrum scan row count | spectrum_scan_row_count | The number of rows of data in Amazon S3 scanned by an Amazon Redshift Spectrum query.<br>Valid values are 0–999,999,999,999,999.                                                                                                               |
| Spectrum scan size      | spectrum_scan_size_mb   | The size of data in Amazon S3, in MB, scanned by an Amazon Redshift Spectrum query.<br>Valid values are 0–999,999,999,999,999.                                                                                                                 |
| Query priority          | query_priority          | The priority of the query.<br>Valid values are HIGHEST, HIGH, NORMAL, LOW, and LOWEST. When comparing query_priority using greater than (>) and less than (<) operators, HIGHEST is greater than HIGH, HIGH is greater than NORMAL, and so on. |

#### Note

- The hop action is not supported with the query\_queue\_time predicate. That is, rules defined to hop when a query\_queue\_time predicate is met are ignored.
- Short segment execution times can result in sampling errors with some metrics, such as io\_skew and query\_cpu\_usage\_percent. To avoid or reduce sampling errors, include segment execution time in your rules. A good starting point is segment\_execution\_time > 10.

The [SVL\\_QUERY\\_METRICS \(p. 1642\)](#) view shows the metrics for completed queries. The [SVL\\_QUERY\\_METRICS\\_SUMMARY \(p. 1643\)](#) view shows the maximum values of metrics for completed queries. Use the values in these views as an aid to determine threshold values for defining query monitoring rules.

## Query monitoring metrics for Amazon Redshift Serverless

The following table describes the metrics used in query monitoring rules for Amazon Redshift Serverless. (These metrics are distinct from the metrics stored in the [STV\\_QUERY\\_METRICS \(p. 1592\)](#) and [STL\\_QUERY\\_METRICS \(p. 1542\)](#) system tables.)

#### Note

The [WLM timeout \(p. 502\)](#) parameter is distinct from query monitoring rules.

| Metric                     | Name                        | Description                                                                                                                                                                                                                                |
|----------------------------|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Query CPU time             | max_query_cpu_time          | CPU time used by the query, in seconds. CPU time is distinct from Query execution time.<br><br>Valid values are 0–999,999.                                                                                                                 |
| Blocks read                | max_query_blocks_read       | Number of 1 MB data blocks read by the query.<br><br>Valid values are 0–1,048,575.                                                                                                                                                         |
| Scan row count             | max_scan_row_count          | The number of rows in a scan step. The row count is the total number of rows emitted before filtering rows marked for deletion (ghost rows) and before applying user-defined query filters.<br><br>Valid values are 0–999,999,999,999,999. |
| Query execution time       | max_query_execution_time    | Elapsed execution time for a query, in seconds. Execution time doesn't include time spent waiting in a queue. If a query exceeds the set execution time, Amazon Redshift Serverless stops the query.<br><br>Valid values are 0–86,399.     |
| Query queue time           | max_query_queue_time        | Time spent waiting in a queue, in seconds.<br><br>Valid values are 0–86,399.                                                                                                                                                               |
| CPU usage                  | max_query_cpu_usage_percent | Percent of CPU capacity used by the query.<br><br>Valid values are 0–6,399.                                                                                                                                                                |
| Memory to disk             | max_query_temp_blocks       | Temporary disk space used to write intermediate results, in 1 MB blocks.<br><br>Valid values are 0–319,815,679.                                                                                                                            |
| Rows joined                | max_join_row_count          | The number of rows processed in a join step.<br><br>Valid values are 0–999,999,999,999,999.                                                                                                                                                |
| Nested loop join row count | max_nested_loop_join        | The number of rows in a nested loop join.<br><br>Valid values are 0–999,999,999,999,999.                                                                                                                                                   |

### Note

- The hop action is not supported with the max\_query\_queue\_time predicate. That is, rules defined to hop when a max\_query\_queue\_time predicate is met are ignored.
- Short segment execution times can result in sampling errors with some metrics, such as max\_io\_skew and max\_query\_cpu\_usage\_percent.

The [SVL\\_QUERY\\_METRICS \(p. 1642\)](#) view shows the metrics for completed queries. The [SVL\\_QUERY\\_METRICS\\_SUMMARY \(p. 1643\)](#) view shows the maximum values of metrics for completed queries. Use the values in these views as an aid to determine threshold values for defining query monitoring rules.

## Query monitoring rules templates

When you add a rule using the Amazon Redshift console, you can choose to create a rule from a predefined template. Amazon Redshift creates a new rule with a set of predicates and populates the predicates with default values. The default action is log. You can modify the predicates and action to meet your use case.

The following table lists available templates.

| Template Name                                     | Predicates                                                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------------------------------------|--------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Nested loop join                                  | <code>nested_loop_join_row &gt; 100</code>                         | A nested loop join might indicate an incomplete join predicate, which often results in a very large return set (a Cartesian product). Use a low row count to find a potentially runaway query early.                                                                                                                                                                                                                                                                                               |
| Query returns a high number of rows               | <code>return_row_count &gt; 1000000</code>                         | If you dedicate a queue to simple, short running queries, you might include a rule that finds queries returning a high row count. The template uses a default of 1 million rows. For some systems, you might consider one million rows to be high, or in a larger system, a billion or more rows might be high.                                                                                                                                                                                    |
| Join with a high number of rows                   | <code>join_row_count &gt; 1000000000</code>                        | A join step that involves an unusually high number of rows might indicate a need for more restrictive filters. The template uses a default of 1 billion rows. For an ad hoc (one-time) queue that's intended for quick, simple queries, you might use a lower number.                                                                                                                                                                                                                              |
| High disk usage when writing intermediate results | <code>query_temp_blocks_to_disk &gt; 100000</code>                 | When currently executing queries use more than the available system RAM, the query execution engine writes intermediate results to disk (spilled memory). Typically, this condition is the result of a rogue query, which usually is also the query that uses the most disk space. The acceptable threshold for disk usage varies based on the cluster node type and number of nodes. The template uses a default of 100,000 blocks, or 100 GB. For a small cluster, you might use a lower number. |
| Long running query with high I/O skew             | <code>segment_execution_time &gt; 120 and io_skew &gt; 1.30</code> | I/O skew occurs when one node slice has a much higher I/O rate than the other slices. As a starting point, a skew of 1.30 (1.3 times average) is considered high. High I/O skew is not always a problem, but when combined with a long running query time, it might indicate a problem with the distribution style or sort key.                                                                                                                                                                    |

## System tables and views for query monitoring rules

When all of a rule's predicates are met, WLM writes a row to the [STL\\_WLM\\_RULE\\_ACTION \(p. 1574\)](#) system table. This row contains details for the query that triggered the rule and the resulting action.

In addition, Amazon Redshift records query metrics the following system tables and views.

- The [STV\\_QUERY\\_METRICS \(p. 1592\)](#) table displays the metrics for currently running queries.
- The [STL\\_QUERY\\_METRICS \(p. 1542\)](#) table records the metrics for completed queries.
- The [SVL\\_QUERY\\_METRICS \(p. 1642\)](#) view shows the metrics for completed queries.
- The [SVL\\_QUERY\\_METRICS\\_SUMMARY \(p. 1643\)](#) view shows the maximum values of metrics for completed queries.

## WLM system tables and views

WLM configures query queues according to WLM service classes, which are internally defined. Amazon Redshift creates several internal queues according to these service classes along with the queues defined in the WLM configuration. The terms *queue* and *service class* are often used interchangeably in the system tables. The superuser queue uses service class 5. User-defined queues use service class 6 and greater.

You can view the status of queries, queues, and service classes by using WLM-specific system tables. Query the following system tables to do the following:

- View which queries are being tracked and what resources are allocated by the workload manager.
- See which queue a query has been assigned to.
- View the status of a query that is currently being tracked by the workload manager.

| Table Name                                              | Description                                                                        |
|---------------------------------------------------------|------------------------------------------------------------------------------------|
| <a href="#">STL_WLM_ERROR (p. 1574)</a>                 | Contains a log of WLM-related error events.                                        |
| <a href="#">STL_WLM_QUERY (p. 1575)</a>                 | Lists queries that are being tracked by WLM.                                       |
| <a href="#">STV_WLM_CLASSIFICATION_CONFIG (p. 1603)</a> | Shows the current classification rules for WLM.                                    |
| <a href="#">STV_WLM_QUERY_QUEUE_STATE (p. 1604)</a>     | Records the current state of the query queues.                                     |
| <a href="#">STV_WLM_QUERY_STATE (p. 1605)</a>           | Provides a snapshot of the current state of queries that are being tracked by WLM. |
| <a href="#">STV_WLM_QUERY_TASK_STATE (p. 1606)</a>      | Contains the current state of query tasks.                                         |
| <a href="#">STV_WLM_SERVICE_CLASS_CONFIG (p. 1608)</a>  | Records the service class configurations for WLM.                                  |
| <a href="#">STV_WLM_SERVICE_CLASS_STATE (p. 1609)</a>   | Contains the current state of the service classes.                                 |

You use the task ID to track a query in the system tables. The following example shows how to obtain the task ID of the most recently submitted user query:

```
select task from stl_wlm_query where exec_start_time =(select max(exec_start_time) from stl_wlm_query);

task

137
(1 row)
```

The following example displays queries that are currently executing or waiting in various service classes (queues). This query is useful in tracking the overall concurrent workload for Amazon Redshift:

```
select * from stv_wlm_query_state order by query;

+-----+-----+-----+-----+-----+-----+-----+
| xid | task|query|service_| wlm_start_ | state |queue_ | exec_
| | | |class | time | |time | time
+-----+-----+-----+-----+-----+-----+-----+
| 2645| 84 | 98 | 3 | 2010-10-... | Returning| 0 | 3438369
| 2650| 85 | 100 | 3 | 2010-10-... | Waiting | 0 | 1645879
| 2660| 87 | 101 | 2 | 2010-10-... | Executing| 0 | 916046
| 2661| 88 | 102 | 1 | 2010-10-... | Executing| 0 | 13291
+-----+-----+-----+-----+-----+-----+-----+
(4 rows)
```

## WLM service class IDs

The following table lists the IDs assigned to service classes.

| ID      | Service class                                                        |
|---------|----------------------------------------------------------------------|
| 1–4     | Reserved for system use.                                             |
| 5       | Used by the superuser queue.                                         |
| 6–13    | Used by manual WLM queues that are defined in the WLM configuration. |
| 14      | Used by short query acceleration.                                    |
| 15      | Reserved for maintenance activities run by Amazon Redshift.          |
| 100–107 | Used by automatic WLM queue when <b>auto_wlm</b> is true.            |

# Managing database security

## Topics

- [Amazon Redshift security overview \(p. 535\)](#)
- [Default database user permissions \(p. 536\)](#)
- [Superusers \(p. 537\)](#)
- [Users \(p. 537\)](#)
- [Groups \(p. 538\)](#)
- [Schemas \(p. 539\)](#)
- [Role-based access control \(RBAC\) \(p. 541\)](#)
- [Row-level security \(p. 547\)](#)
- [Dynamic data masking \(p. 555\)](#)

You manage database security by controlling which users have access to which database objects.

Access to database objects depends on the permissions that you grant to users or groups. The following guidelines summarize how database security works:

- By default, permissions are granted only to the object owner.
- Amazon Redshift database users are named users that can connect to a database. A user is granted permissions in two ways: explicitly, by having those permissions assigned directly to the account, or implicitly, by being a member of a group that is granted permissions.
- Groups are collections of users that can be collectively assigned permissions for streamlined security maintenance.
- Schemas are collections of database tables and other database objects. Schemas are similar to file system directories, except that schemas cannot be nested. Users can be granted access to a single schema or to multiple schemas.

Additionally, Amazon Redshift employs the following features to give you finer control over which users have access to which database objects:

- Role-based access control (RBAC) lets you assign permissions to roles which you can then apply to users, letting you control permissions for large groups of users. Unlike groups, roles can inherit permissions from other roles.

Row-level security (RLS) lets you define policies that restrict access to rows of your choosing, then apply those policies to users or groups.

Dynamic data masking (DDM) further protects your data by transforming it at query runtime so that you can allow users access to data without exposing sensitive details.

For examples of security implementation, see [Example for controlling user and group access \(p. 538\)](#).

For more information about protecting your data, see [Security in Amazon Redshift](#) in the *Amazon Redshift Management Guide*.

## Amazon Redshift security overview

Amazon Redshift database security is distinct from other types of Amazon Redshift security. In addition to database security, which is described in this section, Amazon Redshift provides these features to manage security:

- **Sign-in credentials** — Access to your Amazon Redshift AWS Management Console is controlled by your AWS account permissions. For more information, see [Sign-in credentials](#).
- **Access management** — To control access to specific Amazon Redshift resources, you define AWS Identity and Access Management (IAM) accounts. For more information, see [Controlling access to Amazon Redshift resources](#).
- **Cluster security groups** — To grant other users inbound access to an Amazon Redshift cluster, you define a cluster security group and associate it with a cluster. For more information, see [Amazon Redshift cluster security groups](#).
- **VPC** — To protect access to your cluster by using a virtual networking environment, you can launch your cluster in an Amazon Virtual Private Cloud (VPC). For more information, see [Managing clusters in Virtual Private Cloud \(VPC\)](#).
- **Cluster encryption** — To encrypt the data in all your user-created tables, you can turn on cluster encryption when you launch the cluster. For more information, see [Amazon Redshift clusters](#).
- **SSL connections** — To encrypt the connection between your SQL client and your cluster, you can use secure sockets layer (SSL) encryption. For more information, see [Connect to your cluster using SSL](#).
- **Load data encryption** — To encrypt your table load data files when you upload them to Amazon S3, you can use either server-side encryption or client-side encryption. When you load from server-side encrypted data, Amazon S3 handles decryption transparently. When you load from client-side encrypted data, the Amazon Redshift COPY command decrypts the data as it loads the table. For more information, see [Uploading encrypted data to Amazon S3 \(p. 68\)](#).
- **Data in transit** — To protect your data in transit within the AWS Cloud, Amazon Redshift uses hardware accelerated SSL to communicate with Amazon S3 or Amazon DynamoDB for COPY, UNLOAD, backup, and restore operations.
- **Column-level access control** — To have column-level access control for data in Amazon Redshift, use column-level grant and revoke statements without having to implement views-based access control or use another system.
- **Row-level security control** — To have row-level security control for data in Amazon Redshift, create and attach policies to roles or users that restrict access to rows defined in the policy.

## Default database user permissions

When you create a database object, you are its owner. By default, only a superuser or the owner of an object can query, modify, or grant permissions on the object. For users to use an object, you must grant the necessary permissions to the user or the group that contains the user. Database superusers have the same permissions as database owners.

Amazon Redshift supports the following permissions: SELECT, INSERT, UPDATE, DELETE, REFERENCES, CREATE, TEMPORARY, and USAGE. Different permissions are associated with different object types. For information on database object permissions supported by Amazon Redshift, see the [GRANT \(p. 887\)](#) command.

Only the owner has the permission to modify or destroy an object.

To revoke a permission that was previously granted, use the [REVOKE \(p. 916\)](#) command. The permissions of the object owner, such as DROP, GRANT, and REVOKE permissions, are implicit and cannot be granted or revoked. Object owners can revoke their own ordinary permissions, for example, to make a table read-only for themselves and others. Superusers retain all permissions regardless of GRANT and REVOKE commands.

## Superusers

Database superusers have the same permissions as database owners for all databases.

The *admin* user, which is the user you created when you launched the cluster, is a superuser.

You must be a superuser to create a superuser.

Amazon Redshift system tables and system views are either visible only to superusers or visible to all users. Only superusers can query system tables and system views that are designated "visible to superusers." For information, see [System tables and views \(p. 1409\)](#).

Superusers can view catalog tables. For information, see [System catalog tables \(p. 1670\)](#).

A database superuser bypasses all permission checks. Be careful when using a superuser role. We recommend that you do most of your work as a role that is not a superuser. Superusers retain all permissions regardless of GRANT and REVOKE commands.

To create a new database superuser, log on to the database as a superuser and issue a CREATE USER command or an ALTER USER command with the CREATEUSER permission.

```
CREATE USER adminuser CREATEUSER PASSWORD '1234Admin';
ALTER USER adminuser CREATEUSER;
```

To create, alter, or drop a superuser, use the same commands to manage users. For more information, see [Creating, altering, and deleting users \(p. 537\)](#).

## Users

You can create and manage database users using the Amazon Redshift SQL commands CREATE USER and ALTER USER. Or you can configure your SQL client with custom Amazon Redshift JDBC or ODBC drivers. These manage the process of creating database users and temporary passwords as part of the database logon process.

The drivers authenticate database users based on AWS Identity and Access Management (IAM) authentication. If you already manage user identities outside of AWS, you can use a SAML 2.0-compliant identity provider (IdP) to manage access to Amazon Redshift resources. You use an IAM role to configure your IdP and AWS to permit your federated users to generate temporary database credentials and log on to Amazon Redshift databases. For more information, see [Using IAM authentication to generate database user credentials](#).

Amazon Redshift users can only be created and dropped by a database superuser. Users are authenticated when they log on to Amazon Redshift. They can own databases and database objects (for example, tables). They can also grant permissions on those objects to users, groups, and schemas to control who has access to which object. Users with CREATE DATABASE rights can create databases and grant permissions to those databases. Superusers have database ownership permissions for all databases.

## Creating, altering, and deleting users

Database users are global across a data warehouse cluster (and not for each individual database).

- To create a user, use the [CREATE USER \(p. 846\)](#) command.
- To create a superuser, use the [CREATE USER \(p. 846\)](#) command with the CREATEUSER option.
- To remove an existing user, use the [DROP USER \(p. 876\)](#) command.

- To change a user, for example changing a password, use the [ALTER USER \(p. 659\)](#) command.
- To view a list of users, query the PG\_USER catalog table.

```
select * from pg_user;

 username | usesysid | usecreatedb | usesuper | usecatupd | passwd | valuntil |
 useconfig
-----+-----+-----+-----+-----+-----+
rdsdb | 1 | t | t | t | ***** | *
masteruser | 100 | t | t | f | ***** | *
dwuser | 101 | f | f | f | ***** | *
simpleuser | 102 | f | f | f | ***** | *
poweruser | 103 | f | t | f | ***** | *
dbuser | 104 | t | f | f | ***** | *
(6 rows)
```

## Groups

Groups are collections of users who are all granted whatever permissions are associated with the group. You can use groups to assign permissions by role. For example, you can create different groups for sales, administration, and support and give the users in each group the appropriate access to the data they need for their work. You can grant or revoke permissions at the group level, and those changes will apply to all members of the group, except for superusers.

To view all user groups, query the PG\_GROUP system catalog table:

```
select * from pg_group;
```

## Creating, altering, and deleting groups

Only a superuser can create, alter, or drop groups.

You can perform the following actions:

- To create a group, use the [CREATE GROUP \(p. 792\)](#) command.
- To add users to or remove users from an existing group, use the [ALTER GROUP \(p. 633\)](#) command.
- To delete a group, use the [DROP GROUP \(p. 866\)](#) command. This command only drops the group, not its member users.

## Example for controlling user and group access

This example creates user groups and users and then grants them various permissions for an Amazon Redshift database that connects to a web application client. This example assumes three groups of users: regular users of a web application, power users of a web application, and web developers.

1. Create the groups where the users will be assigned. The following set of commands creates three different user groups:

```
create group webappusers;
create group webpowerusers;
create group webdevusers;
```

2. Create several database users with different permissions and add them to the groups.

a. Create two users and add them to the WEBAPPUSERS group:

```
create user webappuser1 password 'webAppuser1pass'
in group webappusers;

create user webappuser2 password 'webAppuser2pass'
in group webappusers;
```

b. Create a web developer user and add it to the WEBDEVUSERS group:

```
create user webdevuser1 password 'webDevuser2pass'
in group webdevusers;
```

c. Create a superuser. This user will have administrative rights to create other users:

```
create user webappadmin password 'webAppadminpass1'
createuser;
```

3. Create a schema to be associated with the database tables used by the web application, and grant the various user groups access to this schema:

a. Create the WEBAPP schema:

```
create schema webapp;
```

b. Grant USAGE permissions to the WEBAPPUSERS group:

```
grant usage on schema webapp to group webappusers;
```

c. Grant USAGE permissions to the WEBPOWERUSERS group:

```
grant usage on schema webapp to group webpowerusers;
```

d. Grant ALL permissions to the WEBDEVUSERS group:

```
grant all on schema webapp to group webdevusers;
```

The basic users and groups are now set up. You can now alter the users and groups.

4. For example, the following command alters the search\_path parameter for the WEBAPPUSER1.

```
alter user webappuser1 set search_path to webapp, public;
```

The SEARCH\_PATH specifies the schema search order for database objects, such as tables and functions, when the object is referenced by a simple name with no schema specified.

5. You can also add users to a group after creating the group, such as adding WEBAPPUSER2 to the WEBPOWERUSERS group:

```
alter group webpowerusers add user webappuser2;
```

## Schemas

A database contains one or more named schemas. Each schema in a database contains tables and other kinds of named objects. By default, a database has a single schema, which is named PUBLIC. You can

use schemas to group database objects under a common name. Schemas are similar to file system directories, except that schemas cannot be nested.

Identical database object names can be used in different schemas in the same database without conflict. For example, both MY\_SCHEMA and YOUR\_SCHEMA can contain a table named MYTABLE. Users with the necessary permissions can access objects across multiple schemas in a database.

By default, an object is created within the first schema in the search path of the database. For information, see [Search path \(p. 540\)](#) later in this section.

Schemas can help with organization and concurrency issues in a multiuser environment in the following ways:

- To let many developers work in the same database without interfering with each other.
- To organize database objects into logical groups to make them more manageable.
- To give applications the ability to put their objects into separate schemas so that their names will not collide with the names of objects used by other applications.

## Creating, altering, and deleting schemas

Any user can create schemas and alter or drop schemas they own.

You can perform the following actions:

- To create a schema, use the [CREATE SCHEMA \(p. 821\)](#) command.
- To change the owner of a schema, use the [ALTER SCHEMA \(p. 638\)](#) command.
- To delete a schema and its objects, use the [DROP SCHEMA \(p. 872\)](#) command.
- To create a table within a schema, create the table with the format *schema\_name.table\_name*.

To view a list of all schemas, query the PG\_NAMESPACE system catalog table:

```
select * from pg_namespace;
```

To view a list of tables that belong to a schema, query the PG\_TABLE\_DEF system catalog table. For example, the following query returns a list of tables in the PG\_CATALOG schema.

```
select distinct(tablename) from pg_table_def
where schemaname = 'pg_catalog';
```

## Search path

The search path is defined in the search\_path parameter with a comma-separated list of schema names. The search path specifies the order in which schemas are searched when an object, such as a table or function, is referenced by a simple name that does not include a schema qualifier.

If an object is created without specifying a target schema, the object is added to the first schema that is listed in search path. When objects with identical names exist in different schemas, an object name that does not specify a schema will refer to the first schema in the search path that contains an object with that name.

To change the default schema for the current session, use the [SET \(p. 977\)](#) command.

For more information, see the [search\\_path \(p. 1696\)](#) description in the Configuration Reference.

## Schema-based permissions

Schema-based permissions are determined by the owner of the schema:

- By default, all users have CREATE and USAGE permissions on the PUBLIC schema of a database. To disallow users from creating objects in the PUBLIC schema of a database, use the [REVOKE \(p. 916\)](#) command to remove that permission.
- Unless they are granted the USAGE permission by the object owner, users cannot access any objects in schemas they do not own.
- If users have been granted the CREATE permission to a schema that was created by another user, those users can create objects in that schema.

## Role-based access control (RBAC)

By using role-based access control (RBAC) to manage database permissions in Amazon Redshift, you can simplify the management of security permissions in Amazon Redshift. You can secure the access to sensitive data by controlling what users can do both at a broad or fine level. You can also control user access to tasks that are normally restricted to superusers. By assigning different permissions to different roles and assigning them to different users, you can have more granular control of user access.

Users with an assigned role can perform only the tasks that are specified by the assigned role that they are authorized with. For example, a user with the assigned role that has the CREATE TABLE and DROP TABLE permissions is only authorized to perform those tasks. You can control user access by granting different levels of security permissions to different users to access the data they require for their work.

RBAC applies the principle of least permissions to users based on their role requirements, regardless of the types of objects that are involved. Granting and revoking of permissions is performed at the role level, without the need to update permissions on individual database objects.

With RBAC, you can create roles with permissions to run commands that used to require superuser permissions. Users can run these commands, as long as they are authorized with a role that includes these permissions. Similarly, you can also create roles to limit the access to certain commands, and assign the role to either superusers or users that have been authorized with the role.

To learn how Amazon Redshift RBAC works, watch the following video: [Introducing Role-based access control \(RBAC\) in Amazon Redshift](#).

## Role hierarchy

Roles are collections of permissions that you can assign to a user or another role. You can assign system or database permissions to a role. A user inherits permissions from an assigned role.

In RBAC, users can have nested roles. You can grant roles to both users and roles. When granting a role to a user, you authorize the user with all the permissions that this role includes. When granting a role r1 to a user, you authorize the user with permissions from r1. The user now has permissions from r1 and also any existing permissions they already have.

When granting a role (r1) to another role (r2), you authorize r2 with all the permissions from r1. Also, when granting r2 to another role (r3), the permissions of r3 are the combination of the permissions from r1 and r2. Role hierarchy has r2 inherit permissions from r1. Amazon Redshift propagates permissions with each role authorization. Granting r1 to r2 and then r2 to r3 authorizes r3 with all the permissions from the three roles. Thus, by granting r3 to a user, the user has all the permissions from the three roles.

Amazon Redshift doesn't allow the creation of a role authorization cycle. A role authorization cycle happens when a nested role is assigned back to a role earlier in the role hierarchy, such as r3 being assigned back to r1.

## Role assignment

Superusers and regular users with the CREATE ROLE permissions can use the CREATE ROLE statement to create roles. Superusers and role administrators can use the GRANT ROLE statement to grant a role to others. They can use the REVOKE ROLE statement to revoke a role from others, and the DROP ROLE statement to drop roles. Role administrators include role owners and users who have been granted the role with the ADMIN OPTION permission.

Only superusers or role administrators can grant and revoke roles. You can grant or revoke one or more roles to or from one or more roles or users. Use the WITH ADMIN OPTION option in the GRANT ROLE statement to provide the administration options for all the granted roles to all the grantees.

Amazon Redshift supports different combinations of role assignments, such as granting multiple roles or having multiple grantees. The WITH ADMIN OPTION only applies to users and not to roles. Similarly, use the WITH ADMIN OPTION option in the REVOKE ROLE statement to remove the role and the administrative authorization from the grantee. When used with the ADMIN OPTION, only the administrative authorization is revoked from the role.

The following example revokes the administrative authorization of the sample\_role2 role from user2.

```
REVOKE ADMIN OPTION FOR sample_role2 FROM user2;
```

## Amazon Redshift system-defined roles

Amazon Redshift provides a few system-defined roles that are defined with specific permissions. System-specific roles start with a sys: prefix. Only users with appropriate access can alter system-defined roles or create custom system-defined roles. You can't use the sys: prefix for a custom system-defined role.

The following table summarizes the roles and their permissions.

| Role name     | Description                                                                                                                                                                                                                                                                                                                                                               |  |  |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
| sys:monito    | This role has the permission to access catalog or system tables.                                                                                                                                                                                                                                                                                                          |  |  |
| sys:operator  | This role has the permissions to access catalog or system tables, analyze, vacuum, or cancel queries.                                                                                                                                                                                                                                                                     |  |  |
| sys:dba       | This role has the permissions to create schemas, create tables, drop schemas, drop tables, and truncate tables. It has the permissions to create or replace stored procedures, drop procedures, create or replace functions, create or replace external functions, create views, and drop views. Also, this role inherits all the permissions from the sys:operator role. |  |  |
| sys:superuser | This role has all the supported system privileges defined in <a href="#">System permissions for RBAC (p. 543)</a> .                                                                                                                                                                                                                                                       |  |  |
| sys:secadm    | This role has the permissions to create users, alter users, drop users,                                                                                                                                                                                                                                                                                                   |  |  |

| Role name | Description                                                                                                                                     |  |  |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|           | create roles, drop roles, and grant roles. This role can have access to user tables only when the permission is explicitly granted to the role. |  |  |

## System permissions for RBAC

Following is a list of system permissions that you can grant to or revoke from a role.

| Command                  | Minimum permissions required                                                                                                                                                                                                                                                 |  |  |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
| CREATE ROLE              | <ul style="list-style-type: none"> <li>Superuser.</li> <li>Users with the CREATE ROLE permission.</li> </ul>                                                                                                                                                                 |  |  |
| DROP ROLE                | <ul style="list-style-type: none"> <li>Superuser.</li> <li>Role owner who is either the user that created the role or a user that has been granted the role with the WITH ADMIN OPTION permission.</li> </ul>                                                                |  |  |
| CREATE USER              | <ul style="list-style-type: none"> <li>Superuser.</li> <li>Users with the CREATE USER permission. These users can't create superusers.</li> </ul>                                                                                                                            |  |  |
| DROP USER                | <ul style="list-style-type: none"> <li>Superuser.</li> <li>Users with the DROP USER permission.</li> </ul>                                                                                                                                                                   |  |  |
| ALTER USER               | <ul style="list-style-type: none"> <li>Superuser.</li> <li>Users with the ALTER USER permission. These users can't change users to superusers or change superusers to users.</li> <li>Current user who wants to change their own password.</li> </ul>                        |  |  |
| CREATE SCHEMA            | <ul style="list-style-type: none"> <li>Superuser.</li> <li>Users with the CREATE SCHEMA permission.</li> </ul>                                                                                                                                                               |  |  |
| DROP SCHEMA              | <ul style="list-style-type: none"> <li>Superuser.</li> <li>Users with the DROP SCHEMA permission.</li> <li>Schema owner.</li> </ul>                                                                                                                                          |  |  |
| ALTER DEFAULT PRIVILEGES | <ul style="list-style-type: none"> <li>Superuser.</li> <li>Users with the ALTER DEFAULT PRIVILEGES permission.</li> <li>Users changing their own default access permissions.</li> <li>Users setting permissions for schemas that they have access permissions to.</li> </ul> |  |  |
| CREATE TABLE             | <ul style="list-style-type: none"> <li>Superuser.</li> <li>Users with the CREATE TABLE permission.</li> <li>Users with the CREATE permission on schemas.</li> </ul>                                                                                                          |  |  |
| DROP TABLE               | <ul style="list-style-type: none"> <li>Superuser.</li> <li>Users with the DROP TABLE permission.</li> <li>Table owner with the USAGE permission on the schema.</li> </ul>                                                                                                    |  |  |

| <b>Command</b>                      | <b>Minimum permissions required</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |  |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
| ALTER TABLE                         | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the ALTER TABLE permission.</li> <li>• Table owner with the USAGE permission on the schema.</li> </ul>                                                                                                                                                                                                                                                                                                        |  |  |
| CREATE OR REPLACE FUNCTION          | <ul style="list-style-type: none"> <li>• For CREATE FUNCTION:           <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the CREATE OR REPLACE FUNCTION permission.</li> <li>• Users with the USAGE permission on language.</li> </ul> </li> <li>• For REPLACE FUNCTION:           <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the CREATE OR REPLACE FUNCTION permission.</li> <li>• Function owner.</li> </ul> </li> </ul>      |  |  |
| CREATE OR REPLACE EXTERNAL FUNCTION | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the CREATE OR REPLACE EXTERNAL FUNCTION permission.</li> </ul>                                                                                                                                                                                                                                                                                                                                                |  |  |
| DROP FUNCTION                       | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the DROP FUNCTION permission.</li> <li>• Function owner.</li> </ul>                                                                                                                                                                                                                                                                                                                                           |  |  |
| CREATE OR REPLACE PROCEDURE         | <ul style="list-style-type: none"> <li>• For CREATE PROCEDURE:           <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the CREATE OR REPLACE PROCEDURE permission.</li> <li>• Users with the USAGE permission on language.</li> </ul> </li> <li>• For REPLACE PROCEDURE:           <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the CREATE OR REPLACE PROCEDURE permission.</li> <li>• Procedure owner.</li> </ul> </li> </ul> |  |  |
| DROP PROCEDURE                      | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the DROP PROCEDURE permission.</li> <li>• Procedure owner.</li> </ul>                                                                                                                                                                                                                                                                                                                                         |  |  |
| CREATE OR REPLACE VIEW              | <ul style="list-style-type: none"> <li>• For CREATE VIEW:           <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the CREATE OR REPLACE VIEW permission.</li> <li>• Users with the CREATE permission on schemas.</li> </ul> </li> <li>• For REPLACE VIEW:           <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the CREATE OR REPLACE VIEW permission.</li> <li>• View owner.</li> </ul> </li> </ul>                          |  |  |
| DROP VIEW                           | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the DROP VIEW permission.</li> <li>• View owner.</li> </ul>                                                                                                                                                                                                                                                                                                                                                   |  |  |

| Command          | Minimum permissions required                                                                                                                                                                                                                                                                                                                                                                                                             |  |  |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
| CREATE MODEL     | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the CREATE MODEL system permission, who should be able to read the relation of the CREATE MODEL.</li> <li>• Users with the CREATE MODEL permission.</li> </ul>                                                                                                                                                                                                 |  |  |
| DROP MODEL       | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the DROP MODEL permission.</li> <li>• Model owner.</li> <li>• Schema owner.</li> </ul>                                                                                                                                                                                                                                                                         |  |  |
| CREATE DATASHARE | <ul style="list-style-type: none"> <li>• Superuser.</li> </ul> <p>Users with the CREATE DATASHARE permission.</p> <ul style="list-style-type: none"> <li>• Database owner.</li> </ul>                                                                                                                                                                                                                                                    |  |  |
| ALTER DATASHARE  | <ul style="list-style-type: none"> <li>• Superuser.</li> </ul> <p>User with the ALTER DATASHARE permission.</p> <ul style="list-style-type: none"> <li>• Users who have the ALTER or ALL permission on the datashare.</li> <li>• To add specific objects to a datashare, these users must have the permission on the objects. Users should be the owners of objects or have SELECT, USAGE, or ALL permissions on the objects.</li> </ul> |  |  |
| DROP DATASHARE   | <ul style="list-style-type: none"> <li>• Superuser.</li> </ul> <p>Users with the DROP DATASHARE permission.</p> <ul style="list-style-type: none"> <li>• Database owner.</li> </ul>                                                                                                                                                                                                                                                      |  |  |
| CREATE LIBRARY   | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the CREATE LIBRARY permission or with the permission of the specified language.</li> </ul>                                                                                                                                                                                                                                                                     |  |  |
| DROP LIBRARY     | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the DROP LIBRARY permission.</li> <li>• Library owner.</li> </ul>                                                                                                                                                                                                                                                                                              |  |  |
| ANALYZE          | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the ANALYZE permission.</li> <li>• Owner of the relation.</li> <li>• Database owner whom the table is shared to.</li> </ul>                                                                                                                                                                                                                                    |  |  |
| CANCEL           | <ul style="list-style-type: none"> <li>• Superuser canceling their own query.</li> <li>• Superuser canceling a user's query.</li> <li>• Users with the CANCEL permission canceling a user's query.</li> <li>• User canceling their own query.</li> </ul>                                                                                                                                                                                 |  |  |
| TRUNCATE TABLE   | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the TRUNCATE TABLE permission.</li> <li>• Table owner.</li> </ul>                                                                                                                                                                                                                                                                                              |  |  |

| Command     | Minimum permissions required                                                                                                                                                               |  |  |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
| VACUUM      | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users with the VACUUM permission.</li> <li>• Table owner.</li> <li>• Database owner whom the table is shared to.</li> </ul> |  |  |
| IGNORE RLS  | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users within the sys:secadmin role.</li> </ul>                                                                              |  |  |
| EXPLAIN RLS | <ul style="list-style-type: none"> <li>• Superuser.</li> <li>• Users within the sys:secadmin role.</li> </ul>                                                                              |  |  |

## Database object permissions

Apart from system permissions, Amazon Redshift includes database object permissions that define access options. These include such options as the ability to read data in tables and views, write data, create tables, and drop tables. For more information, see [GRANT \(p. 887\)](#) command.

By using RBAC, you can assign database object permissions to roles, similarly to how you can with system permissions. Then you can assign roles to users, authorize users with system permissions, and authorize users with database permissions.

## ALTER DEFAULT PRIVILEGES for RBAC

Use the ALTER DEFAULT PRIVILEGES statement to define the default set of access permissions to be applied to objects that are created in the future by the specified user. By default, users can change only their own default access permissions. With RBAC, you can set the default access permissions for roles. For more information, see the [ALTER DEFAULT PRIVILEGES \(p. 631\)](#) command.

RBAC enables you to assign database object permissions to roles, similarly to system permissions. Then you can assign roles to users, authorize users with system and/or database permissions.

## Considerations for role usage in RBAC

When working with RBAC roles, consider the following:

- Amazon Redshift doesn't allow cycles of role authorizations. You can't grant r1 to r2 and then grant r2 to r1.
- RBAC works for both native Amazon Redshift objects and Amazon Redshift Spectrum tables.
- As an Amazon Redshift administrator, you can turn on RBAC by upgrading your cluster to the latest maintenance patch to get started.
- Only superusers and users with the CREATE ROLE system permission can create roles.
- Only superusers and role administrators can modify or drop roles.
- A role name can't be the same as a user name.
- A role name can't contain invalid characters, such as ":\n."
- A role name can't be a reserved word, such as PUBLIC.
- The role name can't start with the reserved prefix for default roles, sys:.
- You can't drop a role that has the RESTRICT parameter when it is granted to another role. The default setting is RESTRICT. Amazon Redshift throws an error when you try to drop a role that has inherited another role.
- Users that don't have admin permissions on a role can't grant or revoke a role.

## Managing roles in RBAC

To perform the following actions, use the following commands:

- To create a role, use the [CREATE ROLE \(p. 820\)](#) command.
- To rename a role or change the owner of the role, use the [ALTER ROLE \(p. 636\)](#) command.
- To delete a role, use the [DROP ROLE \(p. 871\)](#) command.
- To grant a role to a user, use the [GRANT \(p. 887\)](#) command.
- To revoke a role from a user, use the [REVOKE \(p. 916\)](#) command.
- To grant system permissions to a role, use the [GRANT \(p. 887\)](#) command.
- To revoke system permissions from a role, use the [REVOKE \(p. 916\)](#) command.

## Row-level security

Using row-level security (RLS) in Amazon Redshift, you can have granular access control over your sensitive data. You can decide which users or roles can access specific records of data within schemas or tables, based on security policies that are defined at the database objects level. In addition to column-level security, where you can grant users permissions to a subset of columns, use RLS policies to further restrict access to particular rows of the visible columns. For more information about column-level security, see [Usage notes for column-level access control \(p. 896\)](#).

When you enforce RLS policies on tables, you can restrict returned result sets when users run queries.

When creating RLS policies, you can specify expressions that dictate whether Amazon Redshift returns any existing rows in a table in a query. By creating RLS policies to limit access, you don't have to add or externalize additional conditions in your queries.

When creating RLS policies, we recommend that you create simple policies and avoid complex statements in policies. When defining RLS policies, don't use excessive table joins in the policy definition that are based on policies.

When a policy refers to a lookup table, Amazon Redshift scans the additional table, in addition to the table on which the policy exists. There will be performance differences between the same query for a user with an RLS policy attached, and a user without any policy attached.

## Using RLS policies in SQL statements

When using RLS policies in SQL statements, Amazon Redshift applies the following rules:

- Amazon Redshift applies RLS policies to the SELECT, UPDATE, and DELETE statements by default.
- For SELECT and UNLOAD, Amazon Redshift filters rows according to your defined policy.
- For UPDATE, Amazon Redshift updates only the rows that are visible to you. If a policy restricts a subset of the rows in a table, then you can't update them.
- For DELETE, you can delete only the rows that are visible to you. If a policy restricts a subset of the rows in a table, then you can't delete them. For TRUNCATE, you can still truncate the table.
- For CREATE TABLE LIKE, tables created with the LIKE options won't inherit permission settings from the source table. Similarly, the target table won't inherit the RLS policies from source table.

## Combining multiple policies per user

RLS in Amazon Redshift supports attaching multiple policies per user and object, given that all attached policies have the same table alias. When there are multiple policies defined for a user, Amazon Redshift

applies all the policies with the AND syntax. Multiple policies on a table can be associated with you. Either multiple policies are directly attached to you, or you belong to multiple roles, and the roles have different policies attached to them.

## RLS policy ownership and management

As a superuser, security administrator, or user that has the sys:secadmin role, you can create, modify, or manage all RLS policies for tables. At the object level, you can turn row-level security on or off without modifying the schema definition for tables.

To get started with row-level security, following are SQL statements that you can use:

- Use the ALTER TABLE statement to turn on or off RLS on a table. For more information, see [ALTER TABLE \(p. 640\)](#).
- Use the CREATE RLS POLICY statement to create a security policy for one or more tables, and specify one or more users or roles in the policy.

For more information, see [CREATE RLS POLICY \(p. 819\)](#).

- Use the ATTACH RLS POLICY statement to attach a policy to one or more relations, to one or more users, or to roles.

For more information, see [ATTACH RLS POLICY \(p. 668\)](#).

- Use the DETACH RLS POLICY statement to detach a policy from one or more relations, from one or more users, or from roles.

For more information, see [DETACH RLS POLICY \(p. 862\)](#).

- Use the DROP RLS POLICY statement to drop a policy.

For more information, see [DROP RLS POLICY \(p. 870\)](#).

- Use the GRANT and REVOKE statements to explicitly grant and revoke SELECT permissions to RLS policies that reference lookup tables. For more information, see [GRANT \(p. 887\)](#) and [REVOKE \(p. 916\)](#).

To monitor the policies created, sys:secadmin can view the [SVV\\_RLS\\_POLICY \(p. 1451\)](#) and [SVV\\_RLS\\_ATTACHED\\_POLICY \(p. 1450\)](#).

To list RLS-protected relations, sys:secadmin can view SVV\_RLS\_RELATION.

To trace the application of RLS policies on queries that reference RLS-protected relations, a superuser, sys:operator, or any user with the system permission ACCESS SYSTEM TABLE can view [SVV\\_RLS\\_APPLIED\\_POLICY \(p. 1449\)](#). Note that sys:secadmin is not granted these permissions by default.

To query tables with attached RLS policies, but not see them, you can grant the permission IGNORE RLS to any user. Users that are superusers or sys:secadmin are automatically granted the permission IGNORE RLS. For more information, see [GRANT \(p. 887\)](#).

To explain the RLS policy filters of a query in the EXPLAIN plan to troubleshoot RLS-related queries, you can grant the permission EXPLAIN RLS to any user. For more information, see [GRANT \(p. 887\)](#) and [EXPLAIN \(p. 881\)](#).

## Policy-dependent objects and principles

To provide security for applications and to prevent policy objects from becoming stale or invalid, Amazon Redshift doesn't permit dropping or altering objects referenced by RLS policies.

The following example illustrates how schema dependency is being tracked.

```
-- The CREATE and ATTACH policy statements for `policy_events` references some
-- target and lookup tables.
-- Target tables are tickit_event_redshift and target_schema.target_event_table.
-- Lookup table is tickit_sales_redshift.
-- Policy `policy_events` has following dependencies:
-- table tickit_sales_redshift column eventid, qtysold
-- table tickit_event_redshift column eventid
-- table target_event_table column eventid
-- schema public and target_schema
CREATE RLS POLICY policy_events
WITH (eventid INTEGER)
USING (
 eventid IN (SELECT eventid FROM tickit_sales_redshift WHERE qtysold <3)
);

ATTACH RLS POLICY policy_events ON tickit_event_redshift TO ROLE analyst;

ATTACH RLS POLICY policy_events ON target_schema.target_event_table TO ROLE consumer;
```

Following lists schema object dependencies that Amazon Redshift tracks for RLS policies.

- When tracking schema object dependency for the target table, Amazon Redshift follows these rules:
  - Amazon Redshift detaches the policy from a relation, user, role, or public when you drop a target table.
  - When you rename a target table name, there is no impact to the attached policies.
  - You can only drop the columns of the target table referenced inside the policy definition if you drop or detach the policy first. This also applies when the CASCADE option is specified. You can drop other columns in the target table.
  - You can't rename the referred columns of the target table. To rename referred columns, detach the policy first. This also applies when the CASCADE option is specified.
  - You can't change the type of the referred column, even when you specify the CASCADE option.
- When tracking schema object dependency for the lookup table, Amazon Redshift follows these rules:
  - You can't drop a lookup table. To drop a lookup table, first drop the policy in which the lookup table is referred.
  - You can't rename a lookup table. To rename a lookup table, first drop the policy in which the lookup table is referred. This also applies when the CASCADE option is specified.
  - You can't drop the lookup table columns used in the policy definition. To drop the lookup table columns used in the policy definition, first drop the policy in which the lookup table is referred. This also applies when the CASCADE option is specified in the ALTER TABLE DROP COLUMN statement. You can drop other columns in the lookup table.
  - You can't rename the referred columns of the lookup table. To rename referred columns, first drop the policy in which the lookup table is referred. This also applies when the CASCADE option is specified.
  - You can't change the type of the referred column.
- When a user or role is dropped, Amazon Redshift detaches all policies attached to the user or role automatically.
- When you use the CASCADE option in the DROP SCHEMA statement, Amazon Redshift also drops the relations in the schema. It also drops the relations in any other schemas that are dependent on relations in the dropped schema. For a relation that is a lookup table in a policy, Amazon Redshift fails the DROP SCHEMA DDL. For any relations dropped by the DROP SCHEMA statement, Amazon Redshift detaches all policies that are attached to those relations.
- You can only drop a lookup function (a function that is referred inside a policy definition) when you also drop the policy. This also applies when the CASCADE option is specified.
- When a policy is attached to a table, Amazon Redshift checks if this table is a lookup table in a different policy. If this is the case, Amazon Redshift won't allow attaching a policy to this table.

- While creating an RLS policy, Amazon Redshift checks if this table is a target table for any other RLS policy. If this is the case, Amazon Redshift won't allow creating a policy on this table.

## Considerations

Following are considerations for working with RLS policies:

- Amazon Redshift applies RLS policies to SELECT, UPDATE, or DELETE statements.
- Amazon Redshift doesn't apply RLS policies to INSERT, COPY, ALTER TABLE APPEND statements.
- Row-level security works with column-level security to protect your data.
- When your Amazon Redshift cluster was on the latest generally available version that supports RLS, but is downgraded to an earlier version, Amazon Redshift returns an error when you run a query on base tables with RLS policies attached. The sys:secadmin can revoke access from users that were granted restricted policies, turn off RLS on tables, and drop the policies.
- When RLS is turned on for the source relation, Amazon Redshift supports the ALTER TABLE APPEND statement for superusers, users that have been explicitly granted the system permission IGNORE RLS, or the sys:secadmin role. In this case, you can run the ALTER TABLE APPEND statement to append rows to a target table by moving data from an existing source table. Amazon Redshift moves all tuples from the source relation into the target relation. The RLS status of the target relation doesn't affect the ALTER TABLE APPEND statement.
- To facilitate migration from other data warehouse systems, you can set and retrieve customized session context variables for a connection by specifying the variable name and value.

The following example sets session context variables for a row-level security (RLS) policy.

```
-- Set a customized context variable.
SELECT set_config('app.category', 'Concerts', FALSE);

-- Create a RLS policy using current_setting() to get the value of a customized context
variable.
CREATE RLS POLICY policy_categories
WITH (catgroup VARCHAR(10))
USING (catgroup = current_setting('app.category', FALSE));

-- Set correct roles and attach the policy on the target table to one or more roles.
ATTACH RLS POLICY policy_categories ON ticket_category_redshift TO ROLE analyst, ROLE
dbadmin;
```

For details on how to set and retrieve customized session context variables, see [SET \(p. 977\)](#), [SET\\_CONFIG \(p. 1385\)](#), [SHOW \(p. 981\)](#), [CURRENT\\_SETTING \(p. 1382\)](#), and [RESET \(p. 915\)](#).

- Changing session user using SET SESSION AUTHORIZATION between DECLARE and FETCH or between subsequent FETCH statements won't refresh the already prepared plan based on the user policies at DECLARE time. Avoid changing session user when cursors are used with RLS-protected tables.
- When the base objects inside a view object are RLS-protected, policies attached to the user running the query are applied on the respective base objects. This is different from object-level permission checks, where the view owner's permissions are checked against the view base objects. You can view the RLS-protected relations of a query in its EXPLAIN plan output.
- When a user-defined function (UDF) is referenced in a RLS policy of a relation attached to a user, the user must have the EXECUTE permission over the UDF to query the relation.

## Limitations

Following are the limitations when working with RLS policies:

- Amazon Redshift supports SELECT statements for certain RLS policies with lookups that have complex joins, but doesn't support UPDATE or DELETE statements. In cases with UPDATE or DELETE statements, Amazon Redshift returns the following error:

```
ERROR: One of the RLS policies on target relation is not supported in UPDATE/DELETE.
```

- Whenever a user-defined function (UDF) is referenced in a RLS policy of a relation attached to a user, the user must have the EXECUTE permission over the UDF to query the relation.
- Correlated subqueries aren't supported. Amazon Redshift returns the following error:

```
ERROR: RLS policy could not be rewritten.
```

- RLS policies can't be applied to external tables.
- Amazon Redshift doesn't support datasharing with RLS. If a relation doesn't have RLS turned off for datashares, the query fails on the consumer cluster with the following error:

```
RLS-protected relation "rls_protected_table" cannot be accessed via datasharing query.
```

- In cross-database queries, Amazon Redshift blocks reads to RLS-protected relations. Users with the IGNORE RLS permission can access the protected relation using cross-database queries. When a user without the IGNORE RLS permission accesses RLS-protected relation through a cross-database query, the following error appears:

```
RLS-protected relation "rls_protected_table" cannot be accessed via cross-database query.
```

## Best practices for RLS performance

Following are best practices to ensure better performance from Amazon Redshift on tables protected by RLS.

### Safety of operators and functions

When querying RLS-protected tables, the usage of certain operators or functions may lead to performance degradation. Amazon Redshift classifies operators and functions either as safe or unsafe for querying RLS-protected tables. A function or operator is classified as RLS-safe when it doesn't have any observable side-effects depending on the inputs. In particular, a RLS-safe function or operator can't be one of the following:

- Outputs an input value, or any value that is dependent on the input value, with or without an error message.
- Fails or returns errors that are dependent on the input value.

RLS-unsafe operators include:

- Arithmetic operators — +, -, /, \*, %.
- Text operators — LIKE and SIMILAR TO.
- Cast operators.
- UDFs.

Use the following SELECT statement to check the safety of operators and functions.

```
SELECT proname, proc_is_rls_safe(oid) FROM pg_proc;
```

Amazon Redshift imposes restrictions on the order of evaluation of user predicates containing RLS-unsafe operators and functions when planning queries on RLS-protected tables. Queries referencing RLS-unsafe operators or functions might cause performance degradation when querying RLS-protected tables. Performance can degrade significantly when Amazon Redshift can't push RLS-unsafe predicates down to base table scans to take advantage of sort keys. For better performance, avoid queries using RLS-unsafe predicates that take advantage of a sort key. To verify that Amazon Redshift is able to push down operators and functions, you can use EXPLAIN statements in combination with the system permission EXPLAIN RLS.

## Result caching

To reduce query runtime and improve system performance, Amazon Redshift caches the results of certain types of queries in the memory on the leader node.

Amazon Redshift uses cached results for a new query scanning RLS-protected tables when all the conditions for unprotected tables are true and when all of the following are true:

- The tables or views in the policy haven't been modified.
- The policy doesn't use a function that must be evaluated each time it's run, such as GETDATE or CURRENT\_USER.

For better performance, avoid using policy predicates that don't satisfy the preceding conditions.

For more information on result caching in Amazon Redshift, see [Result caching \(p. 6\)](#).

## Complex policies

For better performance, avoid using complex policies with subqueries that join multiple tables.

# Creating, attaching, detaching, and dropping RLS policies

You can perform the following actions:

- To create an RLS policy, use the [CREATE RLS POLICY \(p. 819\)](#) command.
- To attach an RLS policy on a table to one or more users or roles, use the [ATTACH RLS POLICY \(p. 668\)](#) command.
- To detach a row-level security policy on a table from one or more users or roles, use the [DETACH RLS POLICY \(p. 862\)](#) command.
- To drop an RLS policy for all tables in all databases, use the [DROP RLS POLICY \(p. 870\)](#) command.

The following is an end-to-end example to illustrate how a superuser creates some users and roles. Then, a user with the secadmin role creates, attaches, detaches, and drops RLS policies. This example uses the tickit sample database. For more information, see [Load data from Amazon S3 to Amazon Redshift](#) in the *Amazon Redshift Getting Started Guide*.

```
-- Create users and roles referenced in the policy statements.
CREATE ROLE analyst;
CREATE ROLE consumer;
CREATE ROLE dbadmin;
CREATE ROLE auditor;
CREATE USER bob WITH PASSWORD 'Name_is_bob_1';
CREATE USER alice WITH PASSWORD 'Name_is_alice_1';
CREATE USER joe WITH PASSWORD 'Name_is_joe_1';
```

```

CREATE USER molly WITH PASSWORD 'Name_is_molly_1';
CREATE USER bruce WITH PASSWORD 'Name_is_bruce_1';
GRANT ROLE sys:secadmin TO bob;
GRANT ROLE analyst TO alice;
GRANT ROLE consumer TO joe;
GRANT ROLE dbadmin TO molly;
GRANT ROLE auditor TO bruce;
GRANT ALL ON TABLE tickit_category_redshift TO PUBLIC;
GRANT ALL ON TABLE tickit_sales_redshift TO PUBLIC;
GRANT ALL ON TABLE tickit_event_redshift TO PUBLIC;

-- Create table and schema referenced in the policy statements.
CREATE SCHEMA target_schema;
GRANT ALL ON SCHEMA target_schema TO PUBLIC;
CREATE TABLE target_schema.target_event_table (LIKE tickit_event_redshift);
GRANT ALL ON TABLE target_schema.target_event_table TO PUBLIC;

-- Change session to analyst alice.
SET SESSION AUTHORIZATION alice;

-- Check the tuples visible to analyst alice.
-- Should contain all 3 categories.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Concerts');

SELECT poldb, polname, polalias, polatts, polqual, polenabled, polmodifiedby FROM
svv_rls_policy WHERE poldb = CURRENT_DATABASE();

ATTACH RLS POLICY policy_concerts ON tickit_category_redshift TO ROLE analyst, ROLE
dbadmin;

ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;

SELECT * FROM svv_rls_attached_policy;

-- Change session to analyst alice.
SET SESSION AUTHORIZATION alice;

-- Check that tuples with only `Concert` category will be visible to analyst alice.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to consumer joe.
SET SESSION AUTHORIZATION joe;

-- Although the policy is attached to a different role, no tuples will be
-- visible to consumer joe because the default deny all policy is applied.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to dbadmin molly.
SET SESSION AUTHORIZATION molly;

-- Check that tuples with only `Concert` category will be visible to dbadmin molly.
SELECT catgroup, count(*)
FROM tickit_category_redshift

```

```

GROUP BY catgroup ORDER BY catgroup;

-- Check that EXPLAIN output contains RLS SecureScan to prevent disclosure of
-- sensitive information such as RLS filters.
EXPLAIN SELECT catgroup, count(*) FROM tickit_category_redshift GROUP BY catgroup ORDER BY
catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

-- Grant IGNORE RLS privilege so that RLS policies do not get applicable to role dbadmin.
GRANT IGNORE RLS TO ROLE dbadmin;

-- Grant EXPLAIN RLS privilege so that anyone in role auditor can view complete EXPLAIN
output.
GRANT EXPLAIN RLS TO ROLE auditor;

-- Change session to dbadmin molly.
SET SESSION AUTHORIZATION molly;

-- Check that all tuples are visible to dbadmin molly because `IGNORE RLS` is granted to
role dbadmin.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to auditor bruce.
SET SESSION AUTHORIZATION bruce;

-- Check explain plan is visible to auditor bruce because `EXPLAIN RLS` is granted to role
auditor.
EXPLAIN SELECT catgroup, count(*) FROM tickit_category_redshift GROUP BY catgroup ORDER BY
catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

DETACH RLS POLICY policy_concerts ON tickit_category_redshift FROM ROLE analyst, ROLE
dbadmin;

-- Change session to analyst alice.
SET SESSION AUTHORIZATION alice;

-- Check that no tuples are visible to analyst alice.
-- Although the policy is detached, no tuples will be visible to analyst alice
-- because of default deny all policy is applied if the table has RLS on.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

CREATE RLS POLICY policy_events
WITH (eventid INTEGER) AS ev
USING (
 ev.eventid IN (SELECT eventid FROM tickit_sales_redshift WHERE qtypsold <3)
);

ATTACH RLS POLICY policy_events ON tickit_event_redshift TO ROLE analyst;
ATTACH RLS POLICY policy_events ON target_schema.target_event_table TO ROLE consumer;

RESET SESSION AUTHORIZATION;

-- Can not alter type of dependent column.
ALTER TABLE target_schema.target_event_table ALTER COLUMN eventid TYPE float;

```

```
ALTER TABLE tickit_event_redshift ALTER COLUMN eventid TYPE float;
ALTER TABLE tickit_sales_redshift ALTER COLUMN eventid TYPE float;
ALTER TABLE tickit_sales_redshift ALTER COLUMN qtysold TYPE float;

-- Can not rename dependent column.
ALTER TABLE target_schema.target_event_table RENAME COLUMN eventid TO renamed_eventid;
ALTER TABLE tickit_event_redshift RENAME COLUMN eventid TO renamed_eventid;
ALTER TABLE tickit_sales_redshift RENAME COLUMN eventid TO renamed_eventid;
ALTER TABLE tickit_sales_redshift RENAME COLUMN qtysold TO renamed_qtysold;

-- Can not drop dependent column.
ALTER TABLE target_schema.target_event_table DROP COLUMN eventid CASCADE;
ALTER TABLE tickit_event_redshift DROP COLUMN eventid CASCADE;
ALTER TABLE tickit_sales_redshift DROP COLUMN eventid CASCADE;
ALTER TABLE tickit_sales_redshift DROP COLUMN qtysold CASCADE;

-- Can not drop lookup table.
DROP TABLE tickit_sales_redshift CASCADE;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

DROP RLS POLICY policy_concerts;
DROP RLS POLICY IF EXISTS policy_events;

ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY OFF;

RESET SESSION AUTHORIZATION;

-- Drop users and roles.
DROP USER bob;
DROP USER alice;
DROP USER joe;
DROP USER molly;
DROP USER bruce;
DROP ROLE analyst;
DROP ROLE consumer;
DROP ROLE auditor FORCE;
DROP ROLE dbadmin FORCE;
```

## Dynamic data masking

### Overview

Using dynamic data masking (DDM) in Amazon Redshift, you can protect sensitive data in your data warehouse. You can manipulate how Amazon Redshift shows sensitive data to the user at query time, without transforming it in the database. You control access to data through masking policies that apply custom obfuscation rules to a given user or role. In that way, you can respond to changing privacy requirements without altering underlying data or editing SQL queries.

Dynamic data masking policies hide, obfuscate, or pseudonymize data that matches a given format. When attached to a table, the masking expression is applied to one or more of its columns. You can further modify masking policies to only apply them to certain users, or to user-defined roles that you can create with [Role-based access control \(RBAC\) \(p. 541\)](#). Additionally, you can apply DDM on the cell level by using conditional columns when creating your masking policy. For more information on conditional masking, see [Conditional dynamic data masking \(p. 561\)](#).

You can apply multiple masking policies with varying levels of obfuscation to the same column in a table and assign them to different roles. To avoid conflicts when you have different roles with different

policies applying to one column, you can set priorities for each application. In that way, you can control what data a given user or role can access. DDM policies can partially or completely redact data, or hash it by using user-defined functions written in SQL, Python, or with AWS Lambda. By masking data using hashes, you can apply joins on this data without access to potentially sensitive information.

## End-to-end example

The following is an end-to-end example showing how you can create and attach masking policies to a column. These policies let users access a column and see different values, depending on the degree of obfuscation in the policies attached to their roles. You must be a superuser or have the [sys:secadmin](#) role to run this example.

### Creating a masking policy

First, create a table and populate it with credit card values.

```
--create the table
CREATE TABLE credit_cards (
 customer_id INT,
 credit_card TEXT
);

--populate the table with sample values
INSERT INTO credit_cards
VALUES
 (100, '4532993817514842'),
 (100, '4716002041425888'),
 (102, '5243112427642649'),
 (102, '6011720771834675'),
 (102, '6011378662059710'),
 (103, '373611968625635')
;

--run GRANT to grant permission to use the SELECT statement on the table
GRANT SELECT ON credit_cards TO PUBLIC;

--create two users
CREATE USER regular_user WITH PASSWORD '1234Test!';

CREATE USER analytics_user WITH PASSWORD '1234Test!';

--create the analytics_role role and grant it to analytics_user
--regular_user does not have a role
CREATE ROLE analytics_role;

GRANT ROLE analytics_role TO analytics_user;
```

Next, create a masking policy to apply to the analytics role.

```
--create a masking policy that fully masks the credit card number
CREATE MASKING POLICY mask_credit_card_full
WITH (credit_card VARCHAR(256))
USING ('000000XXXX0000'::TEXT);

--create a user-defined function that partially obfuscates credit card data
CREATE FUNCTION REDACT_CREDIT_CARD (credit_card TEXT)
RETURNS TEXT IMMUTABLE
AS $$
 import re
 regexp = re.compile("^(?P<group1>[0-9]{6})(?P<group2>[0-9]{5,6})(?P<group3>[0-9]{4})$")
```

```

match = regexp.search(credit_card)
if match != None:
 first = match.group(1)
 last = match.group(2)
else:
 first = "000000"
 last = "0000"

return "{}XXXXX{}".format(first, last)
$ LANGUAGE plpythonu;

--create a masking policy that applies the REDACT_CREDIT_CARD function
CREATE MASKING POLICY mask_credit_card_partial
WITH (credit_card VARCHAR(256))
USING (REDACT_CREDIT_CARD(credit_card));

--confirm the masking policies using the associated system views
SELECT * FROM svv_masking_policy;

SELECT * FROM svv_attached_masking_policy;

```

## Attaching a masking policy

Attach the masking policies to the credit card table.

```

--attach mask_credit_card_full to the credit card table as the default policy
--all users will see this masking policy unless a higher priority masking policy is
attached to them or their role
ATTACH MASKING POLICY mask_credit_card_full
ON credit_cards(credit_card)
TO PUBLIC;

--attach mask_credit_card_partial to the analytics role
--users with the analytics role can see partial credit card information
ATTACH MASKING POLICY mask_credit_card_partial
ON credit_cards(credit_card)
TO ROLE analytics_role
PRIORITY 10;

--confirm the masking policies are applied to the table and role in the associated system
view
SELECT * FROM svv_attached_masking_policy;

--confirm the full masking policy is in place for normal users by selecting from the credit
card table as regular_user
SET SESSION AUTHORIZATION regular_user;

SELECT * FROM credit_cards;

--confirm the partial masking policy is in place for users with the analytics role by
selecting from the credit card table as analytics_user
SET SESSION AUTHORIZATION analytics_user;

SELECT * FROM credit_cards;

```

## Altering a masking policy

The following section shows how to alter a dynamic data masking policy.

```
--reset session authorization to the default
RESET SESSION AUTHORIZATION;
```

```
--alter the mask_credit_card_full policy
ALTER MASKING POLICY mask_credit_card_full
USING ('00000000000000'::TEXT);

--confirm the full masking policy is in place after altering the policy, and that results
are altered from '000000XXXX0000' to '00000000000000'
SELECT * FROM credit_cards;
```

## Detaching and dropping a masking policy

The following section shows how to detach and drop masking policies by removing all dynamic data masking policies from the table.

```
--reset session authorization to the default
RESET SESSION AUTHORIZATION;

--detach both masking policies from the credit_cards table
DETACH MASKING POLICY mask_credit_card_full
ON credit_cards(credit_card)
FROM PUBLIC;

DETACH MASKING POLICY mask_credit_card_partial
ON credit_cards(credit_card)
FROM ROLE analytics_role;

--drop both masking policies
DROP MASKING POLICY mask_credit_card_full;

DROP MASKING POLICY mask_credit_card_partial;
```

## Considerations when using dynamic data masking

When using dynamic data masking, consider the following:

- When querying objects created from tables, such as views, users will see results based on their own masking policies, not the policies of the user who created the objects. For example, a user with the analyst role querying a view created by a secadmin would see results with masking policies attached to the analyst role.
- To prevent the EXPLAIN command from potentially exposing sensitive masking policy filters, only users with the SYS\_EXPLAIN\_DDM permission can see masking policies applied in EXPLAIN outputs. Users don't have the SYS\_EXPLAIN\_DDM permission by default.

The following is the syntax for granting the permission to a user or role.

```
GRANT EXPLAIN MASKING TO { username | ROLE rolename }
```

For more information on the EXPLAIN command, see [EXPLAIN \(p. 881\)](#).

- Users with different roles can see differing results based on the filter conditions or join conditions used. For example, running a SELECT command on a table using a specific column value will fail if the user running the command has a masking policy applied that obfuscates that column.
- DDM policies must be applied ahead of any predicate operations, or projections. Masking policies may include the following:
  - Low cost constant operations such as converting a value to null
  - Moderate cost operations such as HMAC hashing
  - High cost operations such as calls to external Lambda user defined functions

As such, we recommend that you use simple masking expressions when possible.

- You can use DDM policies for roles with row-level security policies, but note that RLS policies are applied before DDM. A dynamic data masking expression will not be able to read a row that was protected by RLS. For more information on RLS, see [Row-level security \(p. 547\)](#).
- When using the [COPY \(p. 677\)](#) command to copy from parquet to protected target tables, you should explicitly specify columns in the COPY statement. For more information on mapping columns with COPY, see [Column mapping options \(p. 695\)](#).
- DDM masking policies can attach to lookup tables. They can't attach to the following relations, nor use them in the USING clause:
  - System tables and catalogs
  - External tables
  - Datasharing tables
  - Views, materialized views, and late binding views
  - Cross-DB relations
  - Temporary tables
  - Correlated queries

Following is an example of attaching a masking policy to a lookup table.

```
--Create a masking policy referencing a lookup table
CREATE MASKING POLICY lookup_mask_credit_card WITH (credit_card TEXT) USING (
CASE
 WHEN
 credit_card IN (SELECT credit_card_lookup FROM credit_cards_lookup)
 THEN '000000XXXX0000'
 ELSE REDACT_CREDIT_CARD(credit_card)
END
);

--Provides access to the lookup table via a policy attached to a role
GRANT SELECT ON TABLE credit_cards_lookup TO MASKING POLICY lookup_mask_credit_card;
```

- You can't attach a masking policy that would produce an output incompatible with the target column's type and size. For example, you can't attach a masking policy that outputs a 12 character long string to a VARCHAR(10) column. Amazon Redshift supports the following exceptions:
  - A masking policy with input type INTN can be attached to a policy with size INTM as long as M < N. For example, a BIGINT (INT8) input policy can be attached to a smallint (INT4) column.
  - A masking policy with input type NUMERIC or DECIMAL can always be attached to a FLOAT column.

## Managing dynamic data masking policies

You can perform the following actions:

- To create a DDM policy, use the [CREATE MASKING POLICY \(p. 796\)](#) command.

The following is an example of creating a masking policy using a SHA-2 hash function.

```
CREATE MASKING POLICY hash_credit
WITH (credit_card varchar(256))
USING (sha2(credit_card + 'testSalt', 256));
```

- To alter an existing DDM policy, use the [ALTER MASKING POLICY \(p. 635\)](#) command.

The following is an example of altering an existing masking policy.

```
ALTER MASKING POLICY hash_credit
USING (sha2(credit_card + 'otherTestSalt', 256));
```

- To attach a DDM policy on a table to one or more users or roles, use the [ATTACH MASKING POLICY \(p. 667\)](#) command.

The following is an example of attaching a masking policy to a column/role pair.

```
ATTACH MASKING POLICY hash_credit
ON credit_cards (credit_card)
TO ROLE science_role
PRIORITY 30;
```

The PRIORITY clause determines which masking policy applies to a user session when multiple policies are attached to the same column. For example, if the user in the preceding example has another masking policy attached to the same credit card column with a priority of 20, science\_role's policy is the one that applies, as it has the higher priority of 30.

- To detach a DDM policy on a table from one or more users or roles, use the [DETACH MASKING POLICY \(p. 861\)](#) command.

The following is an example of detaching a masking policy from a column/role pair.

```
DETACH MASKING POLICY hash_credit
ON credit_cards(credit_card)
FROM ROLE science_role;
```

- To drop a DDM policy from all databases, use the [DROP MASKING POLICY \(p. 868\)](#) command.

The following is an example of dropping a masking policy from all databases.

```
DROP MASKING POLICY hash_credit;
```

## Masking policy hierarchy

- You can attach multiple masking policies to a single column.
- When multiple masking policies are applicable to a query, the highest priority policy attached to each respective column applies. Consider the following example.

```
ATTACH MASKING POLICY partial_hash
ON credit_cards(address, credit_card)
TO ROLE analytics_role
PRIORITY 20;

ATTACH MASKING POLICY full_hash
ON credit_cards(credit_card, ssn)
TO ROLE auditor_role
PRIORITY 30;

SELECT address, credit_card, ssn
FROM credit_cards;
```

When running the SELECT statement, a user with both the analytics and auditor roles sees the address column with the partial\_hash masking policy applied. They see the credit card and SSN columns with the full\_hash masking policy applied, because the full\_hash policy has the higher priority on the credit card column.

- If you don't specify a priority when attaching a masking policy, the default priority is 0.
- You can't attach two policies to the same column with equal priority.
- You can't attach two policies to the same column/role pair.

To confirm which masking policy applies to a given role/user and column combination, users with the [sys:secadmin](#) role can look up the column/role or column/user pair in the [SVV\\_ATTACHED\\_MASKING\\_POLICY \(p. 1416\)](#) system view. For more information, see [System views for dynamic data masking \(p. 562\)](#).

## Conditional dynamic data masking

You can mask data at the cell level by creating masking policies with conditional expressions in the masking expression. For example, you can create a masking policy that applies different masks to a value, depending on another column's value in that row.

The following is an example of using conditional data masking to create and attach a masking policy that partially redacts credit card numbers involved in fraud, while completely hiding all other credit card numbers. You must be a superuser or have the [sys:secadmin](#) role to run this example.

```
--Create an analyst role.
CREATE ROLE analyst;

--Create a credit card table. The table contains an is_fraud boolean column,
--which is TRUE if the credit card number in that row was involved in a fraudulent
transaction.
CREATE TABLE credit_cards (id INT, is_fraud BOOLEAN, credit_card_number VARCHAR(16));

--Create a function that partially redacts credit card numbers.
CREATE FUNCTION REDACT_CREDIT_CARD (credit_card VARCHAR(16))
RETURNS VARCHAR(16) IMMUTABLE
AS $$
import re
regexp = re.compile("^(0-9){6}0-9){5,6}(0-9){4})")

match = regexp.search(credit_card)
if match != None:
 first = match.group(1)
 last = match.group(2)
else:
 first = "000000"
 last = "0000"

return "{}XXXXXX{}".format(first, last)
$$ LANGUAGE plpythonu;

--Create a masking policy that partially redacts credit card numbers if the is_fraud value
--for that row is TRUE,
--and otherwise blanks out the credit card number completely.
CREATE MASKING POLICY card_number_conditional_mask
 WITH (fraudulent BOOLEAN, pan varchar(16))
 USING (CASE WHEN fraudulent THEN REDACT_CREDIT_CARD(pan)
 ELSE Null
 END);

--Attach the masking policy to the credit_cards/analyst table/role pair.
ATTACH MASKING POLICY card_number_conditional_mask ON credit_cards (credit_card_number)
 USING (is_fraud, credit_card_number)
 TO ROLE analyst PRIORITY 100;
```

## System views for dynamic data masking

You can access the following DDM related system views with the [sys:secadmin](#) role:

- [SVV\\_MASKING\\_POLICY \(p. 1436\)](#)

Use the SVV\_MASKING\_POLICY to view all masking policies created on the cluster or workgroup.

- [SVV\\_ATTACHED\\_MASKING\\_POLICY \(p. 1416\)](#)

Use the SVV\_ATTACHED\_MASKING\_POLICY to view all the relations and roles/users with policies attached on the currently connected database.

Following are some examples of the information that you can find using system views.

```
--Select all policies associated with specific users, as opposed to roles
SELECT policy_name,
 schema_name,
 table_name,
 grantee
FROM svv_attached_masking_policy
WHERE grantee_type = 'user';

--Select all policies attached to a specific user
SELECT policy_name,
 schema_name,
 table_name,
 grantee
FROM svv_attached_masking_policy
WHERE grantee = 'target_grantee_name';

--Select all policies attached to a given table
SELECT policy_name,
 schema_name,
 table_name,
 grantee
FROM svv_attached_masking_policy
WHERE table_name = 'target_table_name'
 AND schema_name = 'target_schema_name';

--Select the highest priority policy attachment for a given role
SELECT samp.policy_name,
 samp.priority,
 samp.grantee,
 smp.policy_expression
FROM svv_masking_policy AS smp
JOIN svv_attached_masking_policy AS samp
 ON samp.policy_name = smp.policy_name
WHERE
 samp.grantee_type = 'role' AND
 samp.policy_name = mask_get_policy_for_role_on_column(
 'target_schema_name',
 'target_table_name',
 'target_column_name',
 'target_role_name')
ORDER BY samp.priority desc
LIMIT 1;

--See which policy a specific user will see on a specific column in a given relation
SELECT samp.policy_name,
 samp.priority,
 samp.grantee,
 smp.policy_expression
```

```
FROM svv_masking_policy AS smp
JOIN svv_attached_masking_policy AS samp
 ON samp.policy_name = smp.policy_name
WHERE
 samp.grantee_type = 'role' AND
 samp.policy_name = mask_get_policy_for_user_on_column(
 'target_schema_name',
 'target_table_name',
 'target_column_name',
 'target_user_name')
ORDER BY samp.priority desc;
```

# SQL reference

## Topics

- [Amazon Redshift SQL \(p. 564\)](#)
- [Using SQL \(p. 570\)](#)
- [SQL commands \(p. 623\)](#)
- [SQL functions reference \(p. 1020\)](#)
- [Reserved words \(p. 1405\)](#)

## Amazon Redshift SQL

### Topics

- [SQL functions supported on the leader node \(p. 564\)](#)
- [Amazon Redshift and PostgreSQL \(p. 565\)](#)

Amazon Redshift is built around industry-standard SQL, with added functionality to manage very large datasets and support high-performance analysis and reporting of those data.

#### Note

The maximum size for a single Amazon Redshift SQL statement is 16 MB.

## SQL functions supported on the leader node

Some Amazon Redshift queries are distributed and run on the compute nodes, and other queries run exclusively on the leader node.

The leader node distributes SQL to the compute nodes whenever a query references user-created tables or system tables (tables with an STL or STV prefix and system views with an SVL or SVV prefix). A query that references only catalog tables (tables with a PG prefix, such as PG\_TABLE\_DEF, which reside on the leader node) or that does not reference any tables, runs exclusively on the leader node.

Some Amazon Redshift SQL functions are supported only on the leader node and are not supported on the compute nodes. A query that uses a leader-node function must run exclusively on the leader node, not on the compute nodes, or it will return an error.

The documentation for each function that must run exclusively on the leader node includes a note stating that the function will return an error if it references user-defined tables or Amazon Redshift system tables. See [Leader node–only functions \(p. 1020\)](#) for a list of functions that run exclusively on the leader node.

## Examples

The CURRENT\_SCHEMA function is a leader-node only function. In this example, the query does not reference a table, so it runs exclusively on the leader node.

```
select current_schema();
```

The result is as follows.

```
current_schema

public
(1 row)
```

In the next example, the query references a system catalog table, so it runs exclusively on the leader node.

```
select * from pg_table_def
where schemaname = current_schema() limit 1;

schemaname | tablename | column | type | encoding | distkey | sortkey | notnull
-----+-----+-----+-----+-----+-----+-----+-----+
public | category | catid | smallint | none | t | 1 | t
(1 row)
```

In the next example, the query references an Amazon Redshift system table that resides on the compute nodes, so it returns an error.

```
select current_schema(), userid from users;

INFO: Function "current_schema()" not supported.
ERROR: Specified types or functions (one per INFO message) not supported on Amazon
Redshift tables.
```

## Amazon Redshift and PostgreSQL

### Topics

- [Amazon Redshift and PostgreSQL JDBC and ODBC \(p. 566\)](#)
- [Features that are implemented differently \(p. 566\)](#)
- [Unsupported PostgreSQL features \(p. 567\)](#)
- [Unsupported PostgreSQL data types \(p. 568\)](#)
- [Unsupported PostgreSQL functions \(p. 568\)](#)

Amazon Redshift is based on PostgreSQL. Amazon Redshift and PostgreSQL have a number of very important differences that you must be aware of as you design and develop your data warehouse applications.

Amazon Redshift is specifically designed for online analytic processing (OLAP) and business intelligence (BI) applications, which require complex queries against large datasets. Because it addresses very different requirements, the specialized data storage schema and query execution engine that Amazon Redshift uses are completely different from the PostgreSQL implementation. For example, where online transaction processing (OLTP) applications typically store data in rows, Amazon Redshift stores data in columns, using specialized data compression encodings for optimum memory usage and disk I/O. Some PostgreSQL features that are suited to smaller-scale OLTP processing, such as secondary indexes and efficient single-row data manipulation operations, have been omitted to improve performance.

See [System and architecture overview \(p. 2\)](#) for a detailed explanation of the Amazon Redshift data warehouse system architecture.

PostgreSQL 9.x includes some features that are not supported in Amazon Redshift. In addition, there are important differences between Amazon Redshift SQL and PostgreSQL that you must be aware of. This section highlights the differences between Amazon Redshift and PostgreSQL and provides guidance for developing a data warehouse that takes full advantage of the Amazon Redshift SQL implementation.

## Amazon Redshift and PostgreSQL JDBC and ODBC

Because Amazon Redshift is based on PostgreSQL, we previously recommended using JDBC4 Postgresql driver version 8.4.703 and psqlODBC version 9.x drivers. If you are currently using those drivers, we recommend moving to the new Amazon Redshift-specific drivers going forward. For more information about drivers and configuring connections, see [JDBC and ODBC Drivers for Amazon Redshift](#) in the *Amazon Redshift Management Guide*.

To avoid client-side out-of-memory errors when retrieving large data sets using JDBC, you can enable your client to fetch data in batches by setting the JDBC fetch size parameter. For more information, see [Setting the JDBC fetch size parameter \(p. 489\)](#).

Amazon Redshift does not recognize the JDBC maxRows parameter. Instead, specify a [LIMIT \(p. 970\)](#) clause to restrict the result set. You can also use an [OFFSET \(p. 970\)](#) clause to skip to a specific starting point in the result set.

## Features that are implemented differently

Many Amazon Redshift SQL language elements have different performance characteristics and use syntax and semantics and that are quite different from the equivalent PostgreSQL implementation.

### Important

Do not assume that the semantics of elements that Amazon Redshift and PostgreSQL have in common are identical. Make sure to consult the *Amazon Redshift Developer Guide* [SQL commands \(p. 623\)](#) to understand the often subtle differences.

One example in particular is the [VACUUM \(p. 1015\)](#) command, which is used to clean up and reorganize tables. VACUUM functions differently and uses a different set of parameters than the PostgreSQL version. See [Vacuuming tables \(p. 110\)](#) for more about information about using VACUUM in Amazon Redshift.

Often, database management and administration features and tools are different as well. For example, Amazon Redshift maintains a set of system tables and views that provide information about how the system is functioning. See [System tables and views \(p. 1409\)](#) for more information.

The following list includes some examples of SQL features that are implemented differently in Amazon Redshift.

- [CREATE TABLE \(p. 823\)](#)

Amazon Redshift does not support tablespaces, table partitioning, inheritance, and certain constraints. The Amazon Redshift implementation of CREATE TABLE enables you to define the sort and distribution algorithms for tables to optimize parallel processing.

Amazon Redshift Spectrum supports table partitioning using the [CREATE EXTERNAL TABLE \(p. 768\)](#) command.

- [ALTER TABLE \(p. 640\)](#)

Only a subset of ALTER COLUMN actions are supported.

ADD COLUMN supports adding only one column in each ALTER TABLE statement.

- [COPY \(p. 677\)](#)

The Amazon Redshift COPY command is highly specialized to enable the loading of data from Amazon S3 buckets and Amazon DynamoDB tables and to facilitate automatic compression. See the [Loading data \(p. 63\)](#) section and the COPY command reference for details.

- [VACUUM \(p. 1015\)](#)

The parameters for VACUUM are entirely different. For example, the default VACUUM operation in PostgreSQL simply reclaims space and makes it available for re-use; however, the default VACUUM operation in Amazon Redshift is VACUUM FULL, which reclaims disk space and resorts all rows.

- Trailing spaces in VARCHAR values are ignored when string values are compared. For more information, see [Significance of trailing blanks \(p. 584\)](#).

## Unsupported PostgreSQL features

These PostgreSQL features are not supported in Amazon Redshift.

### Important

Do not assume that the semantics of elements that Amazon Redshift and PostgreSQL have in common are identical. Make sure to consult the *Amazon Redshift Developer Guide* [SQL commands \(p. 623\)](#) to understand the often subtle differences.

- The query tool *psql* is unsupported. The [Amazon Redshift RSQL](#) client is supported.
- Table partitioning (range and list partitioning)
- Tablespaces
- Constraints
  - Unique
  - Foreign key
  - Primary key
  - Check constraints
  - Exclusion constraints

Unique, primary key, and foreign key constraints are permitted, but they are informational only. They are not enforced by the system, but they are used by the query planner.

- Database roles
- Inheritance
- Postgres system columns

Amazon Redshift SQL does not implicitly define system columns. However, the PostgreSQL system column names cannot be used as names of user-defined columns. See <https://www.postgresql.org/docs/8.0/static/ddl-system-columns.html>

- Indexes
- NULLS clause in Window functions
- Collations

Amazon Redshift does not support locale-specific or user-defined collation sequences. See [Collation sequences \(p. 601\)](#).

- Value expressions
  - Subscripted expressions
  - Array constructors
  - Row constructors
- Triggers
- Management of External Data (SQL/MED)

- Table functions
- VALUES list used as constant tables
- Sequences
- Full text search

## Unsupported PostgreSQL data types

Generally, if a query attempts to use an unsupported data type, including explicit or implicit casts, it will return an error. However, some queries that use unsupported data types will run on the leader node but not on the compute nodes. See [SQL functions supported on the leader node \(p. 564\)](#).

For a list of the supported data types, see [Data types \(p. 574\)](#).

These PostgreSQL data types are not supported in Amazon Redshift.

- Arrays
- BIT, BIT VARYING
- BYTEA
- Composite Types
- Date/Time Types
  - INTERVAL
- Enumerated Types
- Geometric Types
- HSTORE
- JSON
- Network Address Types
- Numeric Types
  - SERIAL, BIGSERIAL, SMALLSERIAL
  - MONEY
- Object Identifier Types
- Pseudo-Types
- Range Types
- Special Character Types
  - "char" – A single-byte internal type (where the data type named char is enclosed in quotation marks).
  - name – An internal type for object names.

For more information about these types, see [Special Character Types](#) in the PostgreSQL documentation.

- Text Search Types
- TXID\_SNAPSHOT
- UUID
- XML

## Unsupported PostgreSQL functions

Many functions that are not excluded have different semantics or usage. For example, some supported functions will run only on the leader node. Also, some unsupported functions will not return an error

when run on the leader node. The fact that these functions do not return an error in some cases should not be taken to indicate that the function is supported by Amazon Redshift.

**Important**

Do not assume that the semantics of elements that Amazon Redshift and PostgreSQL have in common are identical. Make sure to consult the *Amazon Redshift Database Developer Guide* [SQL commands \(p. 623\)](#) to understand the often subtle differences.

For more information, see [SQL functions supported on the leader node \(p. 564\)](#).

These PostgreSQL functions are not supported in Amazon Redshift.

- Access privilege inquiry functions
- Advisory lock functions
- Aggregate functions
  - STRING\_AGG()
  - ARRAY\_AGG()
  - EVERY()
  - XML\_AGG()
  - CORR()
  - COVAR\_POP()
  - COVAR\_SAMP()
  - REGR\_AVGX(), REGR\_AVGY()
  - REGR\_COUNT()
  - REGR\_INTERCEPT()
  - REGR\_R2()
  - REGR\_SLOPE()
  - REGR\_SXX(), REGR\_SXY(), REGR\_SYY()
- Array functions and operators
- Backup control functions
- Comment information functions
- Database object location functions
- Database object size functions
- Date/Time functions and operators
  - CLOCK\_TIMESTAMP()
  - JUSTIFY\_DAYS(), JUSTIFY\_HOURS(), JUSTIFY\_INTERVAL()
  - PG\_SLEEP()
  - TRANSACTION\_TIMESTAMP()
- ENUM support functions
- Geometric functions and operators
- Generic file access functions
- IS DISTINCT FROM
- Network address functions and operators
- Mathematical functions
  - DIV()
  - SETSEED()
  - WIDTH\_BUCKET()
- Set returning functions

- GENERATE\_SERIES()
- GENERATE\_SUBSCRIPTS()
- Range functions and operators
- Recovery control functions
- Recovery information functions
- ROLLBACK TO SAVEPOINT function
- Schema visibility inquiry functions
- Server signaling functions
- Snapshot synchronization functions
- Sequence manipulation functions
- String functions
  - BIT\_LENGTH()
  - OVERLAY()
  - CONVERT(), CONVERT\_FROM(), CONVERT\_TO()
  - ENCODE()
  - FORMAT()
  - QUOTE\_NONNULLABLE()
  - REGEXP\_MATCHES()
  - REGEXP\_SPLIT\_TO\_ARRAY()
  - REGEXP\_SPLIT\_TO\_TABLE()
- System catalog information functions
- System information functions
  - CURRENT\_CATALOG CURRENT\_QUERY()
  - INET\_CLIENT\_ADDR()
  - INET\_CLIENT\_PORT()
  - INET\_SERVER\_ADDR() INET\_SERVER\_PORT()
  - PG\_CONF\_LOAD\_TIME()
  - PG\_IS\_OTHER\_TEMP\_SCHEMA()
  - PG\_LISTENING\_CHANNELS()
  - PG\_MY\_TEMP\_SCHEMA()
  - PG\_POSTMASTER\_START\_TIME()
  - PG\_TRIGGER\_DEPTH()
  - SHOW VERSION()
- Text search functions and operators
- Transaction IDs and snapshots functions
- Trigger functions
- XML functions

## Using SQL

### Topics

- [SQL reference conventions \(p. 571\)](#)
- [Basic elements \(p. 571\)](#)
- [Expressions \(p. 601\)](#)

- [Conditions \(p. 605\)](#)

The SQL language consists of commands and functions that you use to work with databases and database objects. The language also enforces rules regarding the use of data types, expressions, and literals.

## SQL reference conventions

This section explains the conventions that are used to write the syntax for the SQL expressions, commands, and functions described in the SQL reference section.

| Character      | Description                                                                                                                                                                                                                                                                                                                                                   |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CAPS           | Words in capital letters are key words.                                                                                                                                                                                                                                                                                                                       |
| [ ]            | Brackets denote optional arguments. Multiple arguments in brackets indicate that you can choose any number of the arguments. In addition, arguments in brackets on separate lines indicate that the Amazon Redshift parser expects the arguments to be in the order that they are listed in the syntax. For an example, see <a href="#">SELECT (p. 928)</a> . |
| { }            | Braces indicate that you are required to choose one of the arguments inside the braces.                                                                                                                                                                                                                                                                       |
|                | Pipes indicate that you can choose between the arguments.                                                                                                                                                                                                                                                                                                     |
| <i>italics</i> | Words in italics indicate placeholders. You must insert the appropriate value in place of the word in italics.                                                                                                                                                                                                                                                |
| ...            | An ellipsis indicates that you can repeat the preceding element.                                                                                                                                                                                                                                                                                              |
| '              | Words in single quotation marks indicate that you must type the quotes.                                                                                                                                                                                                                                                                                       |

## Basic elements

### Topics

- [Names and identifiers \(p. 571\)](#)
- [Literals \(p. 573\)](#)
- [Nulls \(p. 573\)](#)
- [Data types \(p. 574\)](#)
- [Collation sequences \(p. 601\)](#)

This section covers the rules for working with database object names, literals, nulls, and data types.

## Names and identifiers

Names identify database objects, including tables and columns, as well as users and passwords. The terms *name* and *identifier* can be used interchangeably. There are two types of identifiers, standard identifiers and quoted or delimited identifiers. Identifiers must consist of only UTF-8 printable characters. ASCII letters in standard and delimited identifiers are case-insensitive and are folded to lowercase in the database. In query results, column names are returned as lowercase by default. To return column names in uppercase, set the [describe\\_field\\_name\\_in\\_uppercase \(p. 1687\)](#) configuration parameter to **true**.

## Standard identifiers

Standard SQL identifiers adhere to a set of rules and must:

- Begin with an ASCII single-byte alphabetic character or underscore character, or a UTF-8 multibyte character two to four bytes long.
- Subsequent characters can be ASCII single-byte alphanumeric characters, underscores, or dollar signs, or UTF-8 multibyte characters two to four bytes long.
- Be between 1 and 127 bytes in length, not including quotation marks for delimited identifiers.
- Contain no quotation marks and no spaces.
- Not be a reserved SQL key word.

## Delimited identifiers

Delimited identifiers (also known as quoted identifiers) begin and end with double quotation marks ("). If you use a delimited identifier, you must use the double quotation marks for every reference to that object. The identifier can contain any standard UTF-8 printable characters other than the double quotation mark itself. Therefore, you can create column or table names that include otherwise illegal characters, such as spaces or the percent symbol.

ASCII letters in delimited identifiers are case-insensitive and are folded to lowercase. To use a double quotation mark in a string, you must precede it with another double quotation mark character.

## Case-sensitive identifiers

Case-sensitive identifiers (also known as mixed-case identifiers) can contain both uppercase and lowercase letters. To use case-sensitive identifiers, you can set the configuration `enable_case_sensitive_identifier` to `true`. You can set this configuration for the cluster or for a session. For more information, see [Default parameter values](#) in the *Amazon Redshift Management Guide* and [enable\\_case\\_sensitive\\_identifier \(p. 1688\)](#).

## Examples

This table shows examples of delimited identifiers, the resulting output, and a discussion:

| Syntax           | Result       | Discussion                                                                                                                                                             |
|------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "group"          | group        | GROUP is a reserved word, so usage of it within an identifier requires double quotation marks.                                                                         |
| ""WHERE""        | "where"      | WHERE is also a reserved word. To include quotation marks in the string, escape each double quotation mark character with additional double quotation mark characters. |
| "This name"      | this name    | Double quotation marks are required to preserve the space.                                                                                                             |
| "This ""IS IT""" | this "is it" | The quotation marks surrounding IS IT must each be preceded by an extra quotation mark in order to become part of the name.                                            |

To create a table named group with a column named this "is it":

```
create table "group" (
```

```
"This ""IS IT"" char(10));
```

The following queries return the same result:

```
select "This ""IS IT"""
from "group";

this "is it"

(0 rows)
```

```
select "this ""is it"""
from "group";

this "is it"

(0 rows)
```

The following fully qualified table.column syntax also returns the same result:

```
select "group"."this ""is it"""
from "group";

this "is it"

(0 rows)
```

The following CREATE TABLE command creates a table with a slash in a column name:

```
create table if not exists city_slash_id(
 "city/id" integer not null,
 state char(2) not null);
```

## Literals

A literal or constant is a fixed data value, composed of a sequence of characters or a numeric constant. Amazon Redshift supports several types of literals, including:

- Numeric literals for integer, decimal, and floating-point numbers. For more information, see [Integer and floating-point literals \(p. 580\)](#).
- Character literals, also referred to as strings, character strings, or character constants
- Datetime and interval literals, used with datetime data types. For more information, see [Date, time, and timestamp literals \(p. 588\)](#) and [Interval literals \(p. 590\)](#).

## Nulls

If a column in a row is missing, unknown, or not applicable, it is a null value or is said to contain null. Nulls can appear in fields of any data type that are not restricted by primary key or NOT NULL constraints. A null is not equivalent to the value zero or to an empty string.

Any arithmetic expression containing a null always evaluates to a null. All operators except concatenation return a null when given a null argument or operand.

To test for nulls, use the comparison conditions IS NULL and IS NOT NULL. Because null represents a lack of data, a null is not equal or unequal to any value or to another null.

## Data types

### Topics

- [Multibyte characters \(p. 575\)](#)
- [Numeric types \(p. 575\)](#)
- [Character types \(p. 582\)](#)
- [Datetime types \(p. 585\)](#)
- [Boolean type \(p. 592\)](#)
- [HLLSKETCH type \(p. 594\)](#)
- [SUPER type \(p. 594\)](#)
- [VARBYTE type \(p. 595\)](#)
- [Type compatibility and conversion \(p. 597\)](#)

Each value that Amazon Redshift stores or retrieves has a data type with a fixed set of associated properties. Data types are declared when tables are created. A data type constrains the set of values that a column or argument can contain.

The following table lists the data types that you can use in Amazon Redshift tables.

| Data type        | Aliases                           | Description                                                |
|------------------|-----------------------------------|------------------------------------------------------------|
| SMALLINT         | INT2                              | Signed two-byte integer                                    |
| INTEGER          | INT, INT4                         | Signed four-byte integer                                   |
| BIGINT           | INT8                              | Signed eight-byte integer                                  |
| DECIMAL          | NUMERIC                           | Exact numeric of selectable precision                      |
| REAL             | FLOAT4                            | Single precision floating-point number                     |
| DOUBLE PRECISION | FLOAT8, FLOAT                     | Double precision floating-point number                     |
| BOOLEAN          | BOOL                              | Logical Boolean (true/false)                               |
| CHAR             | CHARACTER, NCHAR, BPCHAR          | Fixed-length character string                              |
| VARCHAR          | CHARACTER VARYING, NVARCHAR, TEXT | Variable-length character string with a user-defined limit |
| DATE             |                                   | Calendar date (year, month, day)                           |
| TIMESTAMP        | TIMESTAMP WITHOUT TIME ZONE       | Date and time (without time zone)                          |
| TIMESTAMPTZ      | TIMESTAMP WITH TIME ZONE          | Date and time (with time zone)                             |
| GEOMETRY         |                                   | Spatial data                                               |
| GEOGRAPHY        |                                   | Spatial data                                               |
| HLLSKETCH        |                                   | Type used with HyperLogLog sketches.                       |

| Data type | Aliases                   | Description                                                                                                                  |
|-----------|---------------------------|------------------------------------------------------------------------------------------------------------------------------|
| SUPER     |                           | A superset data type that encompasses all scalar types of Amazon Redshift including complex types such as ARRAY and STRUCTS. |
| TIME      | TIME WITHOUT TIME ZONE    | Time of day                                                                                                                  |
| TIMETZ    | TIME WITH TIME ZONE       | Time of day with time zone                                                                                                   |
| VARBYTE   | VARBINARY, BINARY VARYING | Variable-length binary value                                                                                                 |

#### Note

For information about unsupported data types, such as "char" (notice that char is enclosed in quotation marks), see [Unsupported PostgreSQL data types \(p. 568\)](#).

## Multibyte characters

The VARCHAR data type supports UTF-8 multibyte characters up to a maximum of four bytes. Five-byte or longer characters are not supported. To calculate the size of a VARCHAR column that contains multibyte characters, multiply the number of characters by the number of bytes per character. For example, if a string has four Chinese characters, and each character is three bytes long, then you will need a VARCHAR(12) column to store the string.

The VARCHAR data type doesn't support the following invalid UTF-8 codepoints:

0xD800 – 0xDFFF (Byte sequences: ED A0 80 – ED BF BF)

The CHAR data type doesn't support multibyte characters.

## Numeric types

### Topics

- [Integer types \(p. 575\)](#)
- [DECIMAL or NUMERIC type \(p. 576\)](#)
- [Notes about using 128-bit DECIMAL or NUMERIC columns \(p. 577\)](#)
- [Floating-Point types \(p. 577\)](#)
- [Computations with numeric values \(p. 577\)](#)
- [Integer and floating-point literals \(p. 580\)](#)
- [Examples with numeric types \(p. 581\)](#)

Numeric data types include integers, decimals, and floating-point numbers.

### Integer types

Use the SMALLINT, INTEGER, and BIGINT data types to store whole numbers of various ranges. You cannot store values outside of the allowed range for each type.

| Name             | Storage | Range            |
|------------------|---------|------------------|
| SMALLINT or INT2 | 2 bytes | -32768 to +32767 |

| Name                  | Storage | Range                                       |
|-----------------------|---------|---------------------------------------------|
| INTEGER, INT, or INT4 | 4 bytes | -2147483648 to +2147483647                  |
| BIGINT or INT8        | 8 bytes | -9223372036854775808 to 9223372036854775807 |

### DECIMAL or NUMERIC type

Use the DECIMAL or NUMERIC data type to store values with a *user-defined precision*. The DECIMAL and NUMERIC keywords are interchangeable. In this document, *decimal* is the preferred term for this data type. The term *numeric* is used generically to refer to integer, decimal, and floating-point data types.

| Storage                                                  | Range                                                      |
|----------------------------------------------------------|------------------------------------------------------------|
| Variable, up to 128 bits for uncompressed DECIMAL types. | 128-bit signed integers with up to 38 digits of precision. |

Define a DECIMAL column in a table by specifying a *precision* and *scale*:

```
decimal(precision, scale)
```

#### *precision*

The total number of significant digits in the whole value: the number of digits on both sides of the decimal point. For example, the number 48.2891 has a precision of 6 and a scale of 4. The default precision, if not specified, is 18. The maximum precision is 38.

If the number of digits to the left of the decimal point in an input value exceeds the precision of the column minus its scale, the value cannot be copied into the column (or inserted or updated). This rule applies to any value that falls outside the range of the column definition. For example, the allowed range of values for a numeric(5,2) column is -999.99 to 999.99.

#### *scale*

The number of decimal digits in the fractional part of the value, to the right of the decimal point. Integers have a scale of zero. In a column specification, the scale value must be less than or equal to the precision value. The default scale, if not specified, is 0. The maximum scale is 37.

If the scale of an input value that is loaded into a table is greater than the scale of the column, the value is rounded to the specified scale. For example, the PRICEPAID column in the SALES table is a DECIMAL(8,2) column. If a DECIMAL(8,4) value is inserted into the PRICEPAID column, the value is rounded to a scale of 2.

```
insert into sales
values (0, 8, 1, 1, 2000, 14, 5, 4323.8951, 11.00, null);

select pricepaid, salesid from sales where salesid=0;

pricepaid | salesid
-----+-----
4323.90 | 0
(1 row)
```

However, results of explicit casts of values selected from tables are not rounded.

### Note

The maximum positive value that you can insert into a DECIMAL(19,0) column is 9223372036854775807 ( $2^{63} - 1$ ). The maximum negative value is -9223372036854775807.

For example, an attempt to insert the value 999999999999999999 (19 nines) will cause an overflow error. Regardless of the placement of the decimal point, the largest string that Amazon Redshift can represent as a DECIMAL number is 9223372036854775807. For example, the largest value that you can load into a DECIMAL(19,18) column is 9.223372036854775807.

These rules are because DECIMAL values with 19 or fewer significant digits of precision are stored internally as 8-byte integers, while DECIMAL values with 20 to 38 significant digits of precision are stored as 16-byte integers.

### Notes about using 128-bit DECIMAL or NUMERIC columns

Do not arbitrarily assign maximum precision to DECIMAL columns unless you are certain that your application requires that precision. 128-bit values use twice as much disk space as 64-bit values and can slow down query execution time.

### Floating-Point types

Use the REAL and DOUBLE PRECISION data types to store numeric values with *variable precision*. These types are *inexact* types, meaning that some values are stored as approximations, such that storing and returning a specific value may result in slight discrepancies. If you require exact storage and calculations (such as for monetary amounts), use the DECIMAL data type.

REAL represents the single-precision floating point format, according to the IEEE Standard 754 for Binary Floating-Point Arithmetic. It has a precision of about 6 digits, and a range of around 1E-37 to 1E+37. You can also specify this data type as FLOAT4.

DOUBLE PRECISION represents the double-precision floating point format, according to the IEEE Standard 754 for Binary Floating-Point Arithmetic. It has a precision of about 15 digits, and a range of around 1E-307 to 1E+308. You can also specify this data type as FLOAT or FLOAT8.

### Computations with numeric values

In this context, *computation* refers to binary mathematical operations: addition, subtraction, multiplication, and division. This section describes the expected return types for these operations, as well as the specific formula that is applied to determine precision and scale when DECIMAL data types are involved.

When numeric values are computed during query processing, you might encounter cases where the computation is impossible and the query returns a numeric overflow error. You might also encounter cases where the scale of computed values varies or is unexpected. For some operations, you can use explicit casting (type promotion) or Amazon Redshift configuration parameters to work around these problems.

For information about the results of similar computations with SQL functions, see [Aggregate functions \(p. 1022\)](#).

### Return types for computations

Given the set of numeric data types supported in Amazon Redshift, the following table shows the expected return types for addition, subtraction, multiplication, and division operations. The first column on the left side of the table represents the first operand in the calculation, and the top row represents the second operand.

|      | INT2 | INT4 | INT8 | DECIMAL | FLOAT4 | FLOAT8 |
|------|------|------|------|---------|--------|--------|
| INT2 | INT2 | INT4 | INT8 | DECIMAL | FLOAT8 | FLOAT8 |

|                |         |         |         |         |        |        |
|----------------|---------|---------|---------|---------|--------|--------|
| <b>INT4</b>    | INT4    | INT4    | INT8    | DECIMAL | FLOAT8 | FLOAT8 |
| <b>INT8</b>    | INT8    | INT8    | INT8    | DECIMAL | FLOAT8 | FLOAT8 |
| <b>DECIMAL</b> | DECIMAL | DECIMAL | DECIMAL | DECIMAL | FLOAT8 | FLOAT8 |
| <b>FLOAT4</b>  | FLOAT8  | FLOAT8  | FLOAT8  | FLOAT8  | FLOAT4 | FLOAT8 |
| <b>FLOAT8</b>  | FLOAT8  | FLOAT8  | FLOAT8  | FLOAT8  | FLOAT8 | FLOAT8 |

### Precision and scale of computed DECIMAL results

The following table summarizes the rules for computing resulting precision and scale when mathematical operations return DECIMAL results. In this table, p1 and s1 represent the precision and scale of the first operand in a calculation and p2 and s2 represent the precision and scale of the second operand. (Regardless of these calculations, the maximum result precision is 38, and the maximum result scale is 38.)

| Operation | Result precision and scale                                   |
|-----------|--------------------------------------------------------------|
| + or -    | Scale = max(s1, s2)<br>Precision = max(p1-s1, p2-s2)+1+scale |
| *         | Scale = s1+s2<br>Precision = p1+p2+1                         |
| /         | Scale = max(4, s1+p2-s2+1)<br>Precision = p1-s1+ s2+scale    |

For example, the PRICEPAID and COMMISSION columns in the SALES table are both DECIMAL(8,2) columns. If you divide PRICEPAID by COMMISSION (or vice versa), the formula is applied as follows:

```
Precision = 8-2 + 2 + max(4,2+8-2+1)
= 6 + 2 + 9 = 17

Scale = max(4,2+8-2+1) = 9

Result = DECIMAL(17,9)
```

The following calculation is the general rule for computing the resulting precision and scale for operations performed on DECIMAL values with set operators such as UNION, INTERSECT, and EXCEPT or functions such as COALESCE and DECODE:

```
Scale = max(s1,s2)
Precision = min(max(p1-s1,p2-s2)+scale,19)
```

For example, a DEC1 table with one DECIMAL(7,2) column is joined with a DEC2 table with one DECIMAL(15,3) column to create a DEC3 table. The schema of DEC3 shows that the column becomes a NUMERIC(15,3) column.

```
create table dec3 as select * from dec1 union select * from dec2;
```

## Result

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'dec3';

column | type | encoding | distkey | sortkey
-----+-----+-----+-----+
c1 | numeric(15,3) | none | f | 0
```

In the above example, the formula is applied as follows:

```
Precision = min(max(7-2,15-3) + max(2,3), 19)
= 12 + 3 = 15

Scale = max(2,3) = 3

Result = DECIMAL(15,3)
```

## Notes on division operations

For division operations, divide-by-zero conditions return errors.

The scale limit of 100 is applied after the precision and scale are calculated. If the calculated result scale is greater than 100, division results are scaled as follows:

- Precision = precision - (scale - max\_scale)
- Scale = max\_scale

If the calculated precision is greater than the maximum precision (38), the precision is reduced to 38, and the scale becomes the result of: max(38 + scale - precision), min(4, 100))

## Overflow conditions

Overflow is checked for all numeric computations. DECIMAL data with a precision of 19 or less is stored as 64-bit integers. DECIMAL data with a precision that is greater than 19 is stored as 128-bit integers. The maximum precision for all DECIMAL values is 38, and the maximum scale is 37. Overflow errors occur when a value exceeds these limits, which apply to both intermediate and final result sets:

- Explicit casting results in runtime overflow errors when specific data values do not fit the requested precision or scale specified by the cast function. For example, you cannot cast all values from the PRICEPAID column in the SALES table (a DECIMAL(8,2) column) and return a DECIMAL(7,3) result:

```
select pricepaid::decimal(7,3) from sales;
ERROR: Numeric data overflow (result precision)
```

This error occurs because *some* of the larger values in the PRICEPAID column cannot be cast.

- Multiplication operations produce results in which the result scale is the sum of the scale of each operand. If both operands have a scale of 4, for example, the result scale is 8, leaving only 10 digits for the left side of the decimal point. Therefore, it is relatively easy to run into overflow conditions when multiplying two large numbers that both have significant scale.

## Numeric calculations with INTEGER and DECIMAL types

When one of the operands in a calculation has an INTEGER data type and the other operand is DECIMAL, the INTEGER operand is implicitly cast as a DECIMAL:

- INT2 (SMALLINT) is cast as DECIMAL(5,0)
- INT4 (INTEGER) is cast as DECIMAL(10,0)
- INT8 (BIGINT) is cast as DECIMAL(19,0)

For example, if you multiply SALES.COMMISSION, a DECIMAL(8,2) column, and SALES.QTYSOLD, a SMALLINT column, this calculation is cast as:

```
DECIMAL(8,2) * DECIMAL(5,0)
```

### Integer and floating-point literals

Literals or constants that represent numbers can be integer or floating-point.

#### Integer literals

An integer constant is a sequence of the digits 0-9, with an optional positive (+) or negative (-) sign preceding the digits.

#### Syntax

```
[+ | -] digit ...
```

#### Examples

Valid integers include the following:

```
23
-555
+17
```

#### Floating-point literals

Floating-point literals (also referred to as decimal, numeric, or fractional literals) are sequences of digits that can include a decimal point, and optionally the exponent marker (e).

#### Syntax

```
[+ | -] digit ... [.] [digit ...]
[e | E [+ | -] digit ...]
```

#### Arguments

e | E

e or E indicates that the number is specified in scientific notation.

#### Examples

Valid floating-point literals include the following:

```
3.14159
-37.
2.0e19
```

-2E-19

## Examples with numeric types

### CREATE TABLE statement

The following CREATE TABLE statement demonstrates the declaration of different numeric data types:

```
create table film (
 film_id integer,
 language_id smallint,
 original_language_id smallint,
 rental_duration smallint default 3,
 rental_rate numeric(4,2) default 4.99,
 length smallint,
 replacement_cost real default 25.00);
```

### Attempt to insert an integer that is out of range

The following example attempts to insert the value 33000 into a SMALLINT column.

```
insert into film(language_id) values(33000);
```

The range for SMALLINT is -32768 to +32767, so Amazon Redshift returns an error.

```
An error occurred when executing the SQL command:
insert into film(language_id) values(33000)

ERROR: smallint out of range [SQL State=22003]
```

### Insert a decimal value into an integer column

The following example inserts the a decimal value into an INT column.

```
insert into film(language_id) values(1.5);
```

This value is inserted but rounded up to the integer value 2.

### Insert a decimal that succeeds because its scale is rounded

The following example inserts a decimal value that has higher precision than the column.

```
insert into film(rental_rate) values(35.512);
```

In this case, the value 35.51 is inserted into the column.

### Attempt to insert a decimal value that is out of range

In this case, the value 350.10 is out of range. The number of digits for values in DECIMAL columns is equal to the column's precision minus its scale (4 minus 2 for the RENTAL\_RATE column). In other words, the allowed range for a DECIMAL(4,2) column is -99.99 through 99.99.

```
insert into film(rental_rate) values (350.10);
```

```
ERROR: numeric field overflow
DETAIL: The absolute value is greater than or equal to 10^2 for field with precision 4,
scale 2.
```

### Insert variable-precision values into a REAL column

The following example inserts variable-precision values into a REAL column.

```
insert into film(replacement_cost) values(1999.99);
insert into film(replacement_cost) values(19999.99);
select replacement_cost from film;
replacement_cost

20000
1999.99
...
```

The value 19999.99 is converted to 20000 to meet the 6-digit precision requirement for the column. The value 1999.99 is loaded as is.

## Character types

### Topics

- [Storage and ranges \(p. 582\)](#)
- [CHAR or CHARACTER \(p. 583\)](#)
- [VARCHAR or CHARACTER VARYING \(p. 583\)](#)
- [NCHAR and NVARCHAR types \(p. 583\)](#)
- [TEXT and BPCHAR types \(p. 583\)](#)
- [Significance of trailing blanks \(p. 584\)](#)
- [Examples with character types \(p. 584\)](#)

Character data types include CHAR (character) and VARCHAR (character varying).

### Storage and ranges

CHAR and VARCHAR data types are defined in terms of bytes, not characters. A CHAR column can only contain single-byte characters, so a CHAR(10) column can contain a string with a maximum length of 10 bytes. A VARCHAR can contain multibyte characters, up to a maximum of four bytes per character. For example, a VARCHAR(12) column can contain 12 single-byte characters, 6 two-byte characters, 4 three-byte characters, or 3 four-byte characters.

| Name                                    | Storage                                                                         | Range (width of column) |
|-----------------------------------------|---------------------------------------------------------------------------------|-------------------------|
| CHAR, CHARACTER or NCHAR                | Length of string, including trailing blanks (if any)                            | 4096 bytes              |
| VARCHAR, CHARACTER VARYING, or NVARCHAR | 4 bytes + total bytes for characters, where each character can be 1 to 4 bytes. | 65535 bytes (64K -1)    |

| Name   | Storage                              | Range (width of column) |
|--------|--------------------------------------|-------------------------|
| BPCHAR | Converted to fixed-length CHAR(256). | 256 bytes               |
| TEXT   | Converted to VARCHAR(256).           | 260 bytes               |

#### Note

The CREATE TABLE syntax supports the MAX keyword for character data types. For example:

```
create table test(col1 varchar(max));
```

The MAX setting defines the width of the column as 4096 bytes for CHAR or 65535 bytes for VARCHAR.

#### CHAR or CHARACTER

Use a CHAR or CHARACTER column to store fixed-length strings. These strings are padded with blanks, so a CHAR(10) column always occupies 10 bytes of storage.

```
char(10)
```

A CHAR column without a length specification results in a CHAR(1) column.

#### VARCHAR or CHARACTER VARYING

Use a VARCHAR or CHARACTER VARYING column to store variable-length strings with a fixed limit. These strings are not padded with blanks, so a VARCHAR(120) column consists of a maximum of 120 single-byte characters, 60 two-byte characters, 40 three-byte characters, or 30 four-byte characters.

```
varchar(120)
```

If you use the VARCHAR data type without a length specifier in a CREATE TABLE statement, the default length is 256. If used in an expression, the size of the output is determined using the input expression (up to 65535).

#### NCHAR and NVARCHAR types

You can create columns with the NCHAR and NVARCHAR types (also known as NATIONAL CHARACTER and NATIONAL CHARACTER VARYING types). These types are converted to CHAR and VARCHAR types, respectively, and are stored in the specified number of bytes.

An NCHAR column without a length specification is converted to a CHAR(1) column.

An NVARCHAR column without a length specification is converted to a VARCHAR(256) column.

#### TEXT and BPCHAR types

You can create an Amazon Redshift table with a TEXT column, but it is converted to a VARCHAR(256) column that accepts variable-length values with a maximum of 256 characters.

You can create an Amazon Redshift column with a BPCHAR (blank-padded character) type, which Amazon Redshift converts to a fixed-length CHAR(256) column.

## Significance of trailing blanks

Both CHAR and VARCHAR data types store strings up to  $n$  bytes in length. An attempt to store a longer string into a column of these types results in an error, unless the extra characters are all spaces (blanks), in which case the string is truncated to the maximum length. If the string is shorter than the maximum length, CHAR values are padded with blanks, but VARCHAR values store the string without blanks.

Trailing blanks in CHAR values are always semantically insignificant. They are disregarded when you compare two CHAR values, not included in LENGTH calculations, and removed when you convert a CHAR value to another string type.

Trailing spaces in VARCHAR and CHAR values are treated as semantically insignificant when values are compared.

Length calculations return the length of VARCHAR character strings with trailing spaces included in the length. Trailing blanks are not counted in the length for fixed-length character strings.

## Examples with character types

### CREATE TABLE statement

The following CREATE TABLE statement demonstrates the use of VARCHAR and CHAR data types:

```
create table address(
address_id integer,
address1 varchar(100),
address2 varchar(50),
district varchar(20),
city_name char(20),
state char(2),
postal_code char(5)
);
```

The following examples use this table.

### Trailing blanks in variable-length character strings

Because ADDRESS1 is a VARCHAR column, the trailing blanks in the second inserted address are semantically insignificant. In other words, these two inserted addresses *match*.

```
insert into address(address1) values('9516 Magnolia Boulevard');
insert into address(address1) values('9516 Magnolia Boulevard ');
```

```
select count(*) from address
where address1='9516 Magnolia Boulevard';

count

2
(1 row)
```

If the ADDRESS1 column were a CHAR column and the same values were inserted, the COUNT(\*) query would recognize the character strings as the same and return 2.

### Results of the LENGTH function

The LENGTH function recognizes trailing blanks in VARCHAR columns:

```
select length(address1) from address;

length

23
25
(2 rows)
```

A value of Augusta in the CITY\_NAME column, which is a CHAR column, would always return a length of 7 characters, regardless of any trailing blanks in the input string.

### Values that exceed the length of the column

Character strings are not truncated to fit the declared width of the column:

```
insert into address(city_name) values('City of South San Francisco');
ERROR: value too long for type character(20)
```

A workaround for this problem is to cast the value to the size of the column:

```
insert into address(city_name)
values('City of South San Francisco'::char(20));
```

In this case, the first 20 characters of the string (City of South San Fr) would be loaded into the column.

## Datetime types

### Topics

- [Storage and ranges \(p. 585\)](#)
- [DATE \(p. 586\)](#)
- [TIME \(p. 586\)](#)
- [TIMETZ \(p. 586\)](#)
- [TIMESTAMP \(p. 586\)](#)
- [TIMESTAMPTZ \(p. 586\)](#)
- [Examples with datetime types \(p. 587\)](#)
- [Date, time, and timestamp literals \(p. 588\)](#)
- [Interval literals \(p. 590\)](#)

Datetime data types include DATE, TIME, TIMETZ, TIMESTAMP, and TIMESTAMPTZ.

### Storage and ranges

| Name      | Storage | Range                          | Resolution    |
|-----------|---------|--------------------------------|---------------|
| DATE      | 4 bytes | 4713 BC to 294276 AD           | 1 day         |
| TIME      | 8 bytes | 00:00:00 to 24:00:00           | 1 microsecond |
| TIMETZ    | 8 bytes | 00:00:00+1459 to 00:00:00+1459 | 1 microsecond |
| TIMESTAMP | 8 bytes | 4713 BC to 294276 AD           | 1 microsecond |

| Name        | Storage | Range                | Resolution    |
|-------------|---------|----------------------|---------------|
| TIMESTAMPTZ | 8 bytes | 4713 BC to 294276 AD | 1 microsecond |

## DATE

Use the DATE data type to store simple calendar dates without timestamps.

## TIME

TIME is an alias of TIME WITHOUT TIME ZONE.

Use the TIME data type to store the time of day.

TIME columns store values with up to a maximum of six digits of precision for fractional seconds.

By default, TIME values are Coordinated Universal Time (UTC) in both user tables and Amazon Redshift system tables.

## TIMETZ

TIMETZ is an alias of TIME WITH TIME ZONE.

Use the TIMETZ data type to store the time of day with a time zone.

TIMETZ columns store values with up to a maximum of six digits of precision for fractional seconds.

By default, TIMETZ values are UTC in both user tables and Amazon Redshift system tables.

## TIMESTAMP

TIMESTAMP is an alias of TIMESTAMP WITHOUT TIME ZONE.

Use the TIMESTAMP data type to store complete timestamp values that include the date and the time of day.

TIMESTAMP columns store values with up to a maximum of six digits of precision for fractional seconds.

If you insert a date into a TIMESTAMP column, or a date with a partial timestamp value, the value is implicitly converted into a full timestamp value. This full timestamp value has default values (00) for missing hours, minutes, and seconds. Time zone values in input strings are ignored.

By default, TIMESTAMP values are UTC in both user tables and Amazon Redshift system tables.

## TIMESTAMPTZ

TIMESTAMPTZ is an alias of TIMESTAMP WITH TIME ZONE.

Use the TIMESTAMPTZ data type to input complete timestamp values that include the date, the time of day, and a time zone. When an input value includes a time zone, Amazon Redshift uses the time zone to convert the value to UTC and stores the UTC value.

To view a list of supported time zone names, run the following command.

```
select pg_timezone_names();
```

To view a list of supported time zone abbreviations, run the following command.

```
select pg_timezone_abbrevs();
```

You can also find current information about time zones in the [IANA Time Zone Database](#).

The following table has examples of time zone formats.

| Format                    | Example                           |
|---------------------------|-----------------------------------|
| dd mon hh:mi:ss yyyy tz   | 17 Dec 07:37:16 1997 PST          |
| mm/dd/yyyy hh:mi:ss.ss tz | 12/17/1997 07:37:16.00 PST        |
| mm/dd/yyyy hh:mi:ss.ss tz | 12/17/1997 07:37:16.00 US/Pacific |
| yyyy-mm-dd hh:mi:ss+/-tz  | 1997-12-17 07:37:16-08            |
| dd.mm.yyyy hh:mi:ss tz    | 17.12.1997 07:37:16.00 PST        |

TIMESTAMPTZ columns store values with up to a maximum of six digits of precision for fractional seconds.

If you insert a date into a TIMESTAMPTZ column, or a date with a partial timestamp, the value is implicitly converted into a full timestamp value. This full timestamp value has default values (00) for missing hours, minutes, and seconds.

TIMESTAMPTZ values are UTC in user tables.

### Examples with datetime types

Following, you can find examples for working with datetime types supported by Amazon Redshift.

#### Date examples

The following examples insert dates that have different formats and display the output.

```
create table datetable (start_date date, end_date date);
```

```
insert into datetable values ('2008-06-01','2008-12-31');
insert into datetable values ('Jun 1,2008','20081231');
```

```
select * from datetable order by 1;
start_date | end_date

2008-06-01 | 2008-12-31
2008-06-01 | 2008-12-31
```

If you insert a timestamp value into a DATE column, the time portion is ignored and only the date is loaded.

#### Time examples

The following examples insert TIME and TIMETZ values that have different formats and display the output.

```
create table timetable (start_time time, end_time timetz);
```

```
insert into timetable values ('19:11:19','20:41:19 UTC');
insert into timetable values ('191119', '204119 UTC');
```

```
select * from timetable order by 1;
start_time | end_time

19:11:19 | 20:41:19+00
19:11:19 | 20:41:19+00
```

### Time stamp examples

If you insert a date into a `TIMESTAMP` or `TIMESTAMPTZ` column, the time defaults to midnight. For example, if you insert the literal `20081231`, the stored value is `2008-12-31 00:00:00`.

To change the time zone for the current session, use the [SET \(p. 977\)](#) command to set the [timezone \(p. 1700\)](#) configuration parameter.

The following examples insert timestamps that have different formats and display the output.

```
create table tstamp(timeofday timestamp, timeofdaytz timestamptz);

insert into tstamp values('Jun 1,2008 09:59:59', 'Jun 1,2008 09:59:59 EST');
insert into tstamp values('Dec 31,2008 18:20','Dec 31,2008 18:20');
insert into tstamp values('Jun 1,2008 09:59:59 EST', 'Jun 1,2008 09:59:59');
```

```
timeofday

2008-06-01 09:59:59
2008-12-31 18:20:00
(2 rows)
```

### Date, time, and timestamp literals

Following are rules for working with date, time, and timestamp literals supported by Amazon Redshift.

#### Dates

The following input dates are all valid examples of literal date values that you can load into Amazon Redshift tables. The default MDY DateStyle mode is assumed to be in effect. This mode means that the month value precedes the day value in strings such as `1999-01-08` and `01/02/00`.

#### Note

A date or timestamp literal must be enclosed in quotation marks when you load it into a table.

| Input date      | Full date        |
|-----------------|------------------|
| January 8, 1999 | January 8, 1999  |
| 1999-01-08      | January 8, 1999  |
| 1/8/1999        | January 8, 1999  |
| 01/02/00        | January 2, 2000  |
| 2000-Jan-31     | January 31, 2000 |
| Jan-31-2000     | January 31, 2000 |

| <b>Input date</b> | <b>Full date</b>                                                             |
|-------------------|------------------------------------------------------------------------------|
| 31-Jan-2000       | January 31, 2000                                                             |
| 20080215          | February 15, 2008                                                            |
| 080215            | February 15, 2008                                                            |
| 2008.366          | December 31, 2008 (the three-digit part of date must be between 001 and 366) |

### Times

The following input times are all valid examples of literal time values that you can load into Amazon Redshift tables.

| <b>Input times</b> | <b>Description (of time part)</b>                    |
|--------------------|------------------------------------------------------|
| 04:05:06.789       | 4:05 AM and 6.789 seconds                            |
| 04:05:06           | 4:05 AM and 6 seconds                                |
| 04:05              | 4:05 AM exactly                                      |
| 040506             | 4:05 AM and 6 seconds                                |
| 04:05 AM           | 4:05 AM exactly; AM is optional                      |
| 04:05 PM           | 4:05 PM exactly; the hour value must be less than 12 |
| 16:05              | 4:05 PM exactly                                      |

### Timestamps

The following input timestamps are all valid examples of literal time values that you can load into Amazon Redshift tables. All of the valid date literals can be combined with the following time literals.

| <b>Input timestamps (concatenated dates and times)</b> | <b>Description (of time part)</b>                    |
|--------------------------------------------------------|------------------------------------------------------|
| 20080215 04:05:06.789                                  | 4:05 AM and 6.789 seconds                            |
| 20080215 04:05:06                                      | 4:05 AM and 6 seconds                                |
| 20080215 04:05                                         | 4:05 AM exactly                                      |
| 20080215 040506                                        | 4:05 AM and 6 seconds                                |
| 20080215 04:05 AM                                      | 4:05 AM exactly; AM is optional                      |
| 20080215 04:05 PM                                      | 4:05 PM exactly; the hour value must be less than 12 |
| 20080215 16:05                                         | 4:05 PM exactly                                      |
| 20080215                                               | Midnight (by default)                                |

## Special datetime values

The following special values can be used as datetime literals and as arguments to date functions. They require single quotation marks and are converted to regular timestamp values during query processing.

| Special value | Description                                                                                                |
|---------------|------------------------------------------------------------------------------------------------------------|
| now           | Evaluates to the start time of the current transaction and returns a timestamp with microsecond precision. |
| today         | Evaluates to the appropriate date and returns a timestamp with zeroes for the time parts.                  |
| tomorrow      | Evaluates to the appropriate date and returns a timestamp with zeroes for the time parts.                  |
| yesterday     | Evaluates to the appropriate date and returns a timestamp with zeroes for the time parts.                  |

The following examples show how now and today work with the DATEADD function.

```
select dateadd(day,1,'today');

date_add

2009-11-17 00:00:00
(1 row)

select dateadd(day,1,'now');

date_add

2009-11-17 10:45:32.021394
(1 row)
```

## Interval literals

Following are rules for working with interval literals supported by Amazon Redshift.

Use an interval literal to identify specific periods of time, such as 12 hours or 6 weeks. You can use these interval literals in conditions and calculations that involve datetime expressions.

### Note

You can't use the INTERVAL data type for columns in Amazon Redshift tables.

An interval is expressed as a combination of the INTERVAL keyword with a numeric quantity and a supported date part, for example INTERVAL '7 days' or INTERVAL '59 minutes'. You can connect several quantities and units to form a more precise interval, for example: INTERVAL '7 days, 3 hours, 59 minutes'. Abbreviations and plurals of each unit are also supported; for example: 5 s, 5 second, and 5 seconds are equivalent intervals.

If you don't specify a date part, the interval value represents seconds. You can specify the quantity value as a fraction (for example: 0.5 days).

## Examples

The following examples show a series of calculations with different interval values.

The following adds 1 second to the specified date.

```
select caldate + interval '1 second' as dateplus from date
where caldate='12-31-2008';
dateplus

2008-12-31 00:00:01
(1 row)
```

The following adds 1 minute to the specified date.

```
select caldate + interval '1 minute' as dateplus from date
where caldate='12-31-2008';
dateplus

2008-12-31 00:01:00
(1 row)
```

The following adds 3 hours and 35 minutes to the specified date.

```
select caldate + interval '3 hours, 35 minutes' as dateplus from date
where caldate='12-31-2008';
dateplus

2008-12-31 03:35:00
(1 row)
```

The following adds 52 weeks to the specified date.

```
select caldate + interval '52 weeks' as dateplus from date
where caldate='12-31-2008';
dateplus

2009-12-30 00:00:00
(1 row)
```

The following adds 1 week, 1 hour, 1 minute, and 1 second to the specified date.

```
select caldate + interval '1w, 1h, 1m, 1s' as dateplus from date
where caldate='12-31-2008';
dateplus

2009-01-07 01:01:01
(1 row)
```

The following adds 12 hours (half a day) to the specified date.

```
select caldate + interval '0.5 days' as dateplus from date
where caldate='12-31-2008';
dateplus

2008-12-31 12:00:00
(1 row)
```

The following subtracts 4 months from February 15, 2023 and the result is October 15, 2022.

```
select date '2023-02-15' - interval '4 months';
```

```
?column?

2022-10-15 00:00:00
```

The following subtracts 4 months from March 31, 2023 and the result is November 30, 2022. The calculation considers the number of days in a month.

```
select date '2023-03-31' - interval '4 months';
?column?

2022-11-30 00:00:00
```

## Boolean type

Use the BOOLEAN data type to store true and false values in a single-byte column. The following table describes the three possible states for a Boolean value and the literal values that result in that state. Regardless of the input string, a Boolean column stores and outputs "t" for true and "f" for false.

| State   | Valid literal values                 | Storage |
|---------|--------------------------------------|---------|
| True    | TRUE 't'<br>'true' 'y'<br>'yes' '1'  | 1 byte  |
| False   | FALSE 'f'<br>'false' 'n'<br>'no' '0' | 1 byte  |
| Unknown | NULL                                 | 1 byte  |

You can use an IS comparison to check a Boolean value only as a predicate in the WHERE clause. You can't use the IS comparison with a Boolean value in the SELECT list.

### Examples

You could use a BOOLEAN column to store an "Active/Inactive" state for each customer in a CUSTOMER table.

```
create table customer(
 custid int,
 active_flag boolean default true);
```

```
insert into customer values(100, default);
```

```
select * from customer;
custid | active_flag
-----+-----
 100 | t
```

If no default value (true or false) is specified in the CREATE TABLE statement, inserting a default value means inserting a null.

In this example, the query selects users from the USERS table who like sports but do not like theatre:

```
select firstname, lastname, likesports, liketheatre
from users
where likesports is true and liketheatre is false
order by userid limit 10;

firstname | lastname | likesports | liketheatre
-----+-----+-----+
Lars | Ratliff | t | f
Mufutau | Watkins | t | f
Scarlett | Mayer | t | f
Shafira | Glenn | t | f
Winifred | Cherry | t | f
Chase | Lamb | t | f
Liberty | Ellison | t | f
Aladdin | Haney | t | f
Tashya | Michael | t | f
Lucian | Montgomery | t | f
(10 rows)
```

The following example selects users from the USERS table for whom it is unknown whether they like rock music.

```
select firstname, lastname, likerock
from users
where likerock is unknown
order by userid limit 10;

firstname | lastname | likerock
-----+-----+
Rafael | Taylor |
Vladimir | Humphrey |
Barry | Roy |
Tamekah | Juarez |
Mufutau | Watkins |
Naida | Calderon |
Anika | Huff |
Bruce | Beck |
Mallory | Farrell |
Scarlett | Mayer |
(10 rows)
```

The following example returns an error because it uses an IS comparison in the SELECT list.

```
select firstname, lastname, likerock is true as "check"
from users
order by userid limit 10;

[Amazon](500310) Invalid operation: Not implemented
```

The following example succeeds because it uses an equal comparison ( = ) in the SELECT list instead of the IS comparison.

```
select firstname, lastname, likerock = true as "check"
from users
order by userid limit 10;

firstname | lastname | check
-----+-----+-----+
Rafael | Taylor |
```

|          |           |       |
|----------|-----------|-------|
| Vladimir | Humphrey  |       |
| Lars     | Ratliff   | true  |
| Barry    | Roy       |       |
| Reagan   | Hodge     | true  |
| Victor   | Hernandez | true  |
| Tamekah  | Juarez    |       |
| Colton   | Roy       | false |
| Mufutau  | Watkins   |       |
| Naida    | Calderon  |       |

## HLLSKETCH type

Use the HLLSKETCH data type for HyperLogLog sketches. Amazon Redshift supports HyperLogLog sketch representations that are either sparse or dense. Sketches begin as sparse and switch to dense when the dense format is more efficient to minimize the memory footprint that is used.

Amazon Redshift automatically transitions a sparse HyperLogLog sketch when importing, exporting, or printing sketches in the following JSON format.

```
{"logm":15,"sparse":{"indices":[4878,9559,14523],"values":[1,2,1]}}
```

Amazon Redshift uses a string representation in a Base64 format to represent a dense HyperLogLog sketch.

Amazon Redshift uses the following string representation in a Base64 format to represent a dense HyperLogLog sketch.

```
"ABAABA..."
```

The maximum size of a HLLSKETCH object is 24,580 bytes when used in raw compression.

## SUPER type

Use the SUPER data type to store semistructured data or documents as values.

Semistructured data doesn't conform to the rigid and tabular structure of the relational data model used in SQL databases. It contains tags that reference distinct entities within the data. They can contain complex values such as arrays, nested structures, and other complex structures that are associated with serialization formats, such as JSON. The SUPER data type is a set of schemaless array and structure values that encompass all other scalar types of Amazon Redshift.

The SUPER data type supports up to 1 MB of data for an individual SUPER field or object. For more information on the SUPER data type, including examples of implementing it in a table, see [Ingesting and querying semistructured data in Amazon Redshift \(p. 368\)](#). On preview clusters and workgroups, the SUPER data type supports up to 16 MB of data for an individual SUPER field or object. For more information on the large SUPER data type, see [Limitations \(p. 388\)](#).

The SUPER data type has the following properties:

- An Amazon Redshift scalar value:
  - A null
  - A boolean
  - A number, such as smallint, integer, bigint, decimal, or floating point (such as float4 or float8)
  - A string value, such as varchar or char
- A complex value:
  - An array of values, including scalar or complex

- A structure, also known as tuple or object, that is a map of attribute names and values (scalar or complex)

Any of the two types of complex values contain their own scalars or complex values without having any restrictions for regularity.

The SUPER data type supports the persistence of semistructured data in a schemaless form. Although hierarchical data model can change, the old versions of data can coexist in the same SUPER column.

For information about how Amazon Redshift uses PartiQL to enable navigation into arrays and structures, see [Navigation \(p. 377\)](#). For information about how Amazon Redshift uses the PartiQL syntax to iterate over SUPER arrays by navigating the array using the FROM clause of a query, see [Unnesting queries \(p. 378\)](#).

For information about how Amazon Redshift uses dynamic typing to process schemaless SUPER data without the need to declare the data types before you use them in your query, see [Dynamic typing \(p. 380\)](#).

## VARBYTE type

Use a VARBYTE, VARBINARY, or BINARY VARYING column to store variable-length binary value with a fixed limit.

```
varbyte [(n)]
```

The maximum number of bytes (*n*) can range from 1 – 1,024,000. The default is 64,000.

Some examples where you might want to use a VARBYTE data type are as follows:

- Joining tables on VARBYTE columns.
- Creating materialized views that contain VARBYTE columns. Incremental refresh of materialized views that contain VARBYTE columns is supported. However, aggregate functions other than COUNT, MIN, and MAX and GROUP BY on VARBYTE columns don't support incremental refresh.

To ensure that all bytes are printable characters, Amazon Redshift uses the hex format to print VARBYTE values. For example, the following SQL converts the hexadecimal string 6162 into a binary value. Even though the returned value is a binary value, the results are printed as hexadecimal 6162.

```
select from_hex('6162');

from_hex

6162
```

Amazon Redshift supports casting between VARBYTE and the following data types:

- CHAR
- VARCHAR
- SMALLINT
- INTEGER
- BIGINT

When casting with CHAR and VARCHAR the UTF-8 format is used. For more information about the UTF-8 format, see [TO\\_VARBYTE \(p. 1049\)](#). When casting from SMALLINT, INTEGER, and BIGINT the number of

bytes of the original data type is maintained. That is two bytes for SMALLINT, four bytes for INTEGER, and eight bytes for BIGINT.

The following SQL statement casts a VARCHAR string to a VARBYTE. Even though the returned value is a binary value, the results are printed as hexadecimal 616263.

```
select 'abc'::varbyte;

varbyte

616263
```

The following SQL statement casts a CHAR value in a column to a VARBYTE. This example creates a table with a CHAR(10) column (c), inserts character values that are shorter than the length of 10. The resulting cast pads the result with a space characters (hex'20') to the defined column size. Even though the returned value is a binary value, the results are printed as hexadecimal.

```
create table t (c char(10));
insert into t values ('aa'), ('abc');
select c::varbyte from t;

c

616120202020202020
61626320202020202020
```

The following SQL statement casts a SMALLINT string to a VARBYTE. Even though the returned value is a binary value, the results are printed as hexadecimal 0005, which is two bytes or four hexadecimal characters.

```
select 5::smallint::varbyte;

varbyte

0005
```

The following SQL statement casts an INTEGER to a VARBYTE. Even though the returned value is a binary value, the results are printed as hexadecimal 00000005, which is four bytes or eight hexadecimal characters.

```
select 5::int::varbyte;

varbyte

00000005
```

The following SQL statement casts a BIGINT to a VARBYTE. Even though the returned value is a binary value, the results are printed as hexadecimal 0000000000000005, which is eight bytes or 16 hexadecimal characters.

```
select 5::bigint::varbyte;

varbyte

0000000000000005
```

Amazon Redshift features that support the VARBYTE data type include:

- [VARBYTE operators \(p. 1045\)](#)

- [CONCAT \(p. 1286\)](#)
- [LEN \(p. 1292\)](#)
- [LENGTH function \(p. 1293\)](#)
- [OCTET\\_LENGTH \(p. 1297\)](#)
- [SUBSTR function \(p. 1318\)](#)
- [SUBSTRING function \(p. 1318\)](#)
- [FROM\\_HEX \(p. 1046\)](#)
- [TO\\_HEX \(p. 1048\)](#)
- [FROM\\_VARBYTE \(p. 1047\)](#)
- [TO\\_VARBYTE \(p. 1049\)](#)
- [GETBIT \(p. 1048\)](#)
- [Loading a column of the VARBYTE data type \(p. 719\)](#)
- [Unloading a column of the VARBYTE data type \(p. 998\)](#)

### Limitations when using the VARBYTE data type with Amazon Redshift

The following are limitations when using the VARBYTE data type with Amazon Redshift:

- Amazon Redshift Spectrum supports the VARBYTE data type only for Parquet and ORC files.
- Amazon Redshift query editor and Amazon Redshift query editor v2 don't yet fully support VARBYTE data type. Therefore, use a different SQL client when working with VARBYTE expressions.

As a workaround to use the query editor, if the length of your data is below 64 KB and the content is valid UTF-8, you can cast the VARBYTE values to VARCHAR, for example:

```
select to_varbyte('6162', 'hex')::varchar;
```

- You can't use VARBYTE data types with Python or Lambda user-defined functions (UDFs).
- You can't create a HLLSKETCH column from a VARBYTE column or use APPROXIMATE COUNT DISTINCT on a VARBYTE column.

## Type compatibility and conversion

Following, you can find a discussion about how type conversion rules and data type compatibility work in Amazon Redshift.

### Compatibility

Data type matching and matching of literal values and constants to data types occurs during various database operations, including the following:

- Data manipulation language (DML) operations on tables
- UNION, INTERSECT, and EXCEPT queries
- CASE expressions
- Evaluation of predicates, such as LIKE and IN
- Evaluation of SQL functions that do comparisons or extractions of data
- Comparisons with mathematical operators

The results of these operations depend on type conversion rules and data type compatibility.

*Compatibility* implies that a one-to-one matching of a certain value and a certain data type is not always required. Because some data types are *compatible*, an implicit conversion, or *coercion*, is possible (for

more information, see [Implicit conversion types \(p. 598\)](#)). When data types are incompatible, you can sometimes convert a value from one data type to another by using an explicit conversion function.

### General compatibility and conversion rules

Note the following compatibility and conversion rules:

- In general, data types that fall into the same type category (such as different numeric data types) are compatible and can be implicitly converted.

For example, with implicit conversion you can insert a decimal value into an integer column. The decimal is rounded to produce a whole number. Or you can extract a numeric value, such as 2008, from a date and insert that value into an integer column.

- Numeric data types enforce overflow conditions that occur when you attempt to insert out-of-range values. For example, a decimal value with a precision of 5 does not fit into a decimal column that was defined with a precision of 4. An integer or the whole part of a decimal is never truncated; however, the fractional part of a decimal can be rounded up or down, as appropriate. However, results of explicit casts of values selected from tables are not rounded.
- Different types of character strings are compatible; VARCHAR column strings containing single-byte data and CHAR column strings are comparable and implicitly convertible. VARCHAR strings that contain multibyte data are not comparable. Also, you can convert a character string to a date, time, timestamp, or numeric value if the string is an appropriate literal value; any leading or trailing spaces are ignored. Conversely, you can convert a date, time, timestamp, or numeric value to a fixed-length or variable-length character string.

#### Note

A character string that you want to cast to a numeric type must contain a character representation of a number. For example, you can cast the strings '1.0' or '5.9' to decimal values, but you cannot cast the string 'ABC' to any numeric type.

- If you compare DECIMAL values with character strings, Amazon Redshift attempts to convert the character string to a DECIMAL value. When comparing all other numeric values with character strings, the numeric values are converted to character strings. To enforce the opposite conversion (for example, converting character strings to integers, or converting DECIMAL values to character strings), use an explicit function, such as [CAST \(p. 1066\)](#).
- To convert 64-bit DECIMAL or NUMERIC values to a higher precision, you must use an explicit conversion function such as the CAST or CONVERT functions.
- When converting DATE or TIMESTAMP to Timestamptz, or converting TIME to Timetz, the time zone is set to the current session time zone. The session time zone is UTC by default. For more information about setting the session time zone, see [timezone \(p. 1700\)](#).
- Similarly, Timestamptz is converted to DATE, TIME, or TIMESTAMP based on the current session time zone. The session time zone is UTC by default. After the conversion, time zone information is dropped.
- Character strings that represent a timestamp with time zone specified are converted to Timestamptz using the current session time zone, which is UTC by default. Likewise, character strings that represent a time with time zone specified are converted to Timetz using the current session time zone, which is UTC by default.

### Implicit conversion types

There are two types of implicit conversions:

- Implicit conversions in assignments, such as setting values in INSERT or UPDATE commands.
- Implicit conversions in expressions, such as performing comparisons in the WHERE clause.

The table following lists the data types that can be converted implicitly in assignments or expressions. You can also use an explicit conversion function to perform these conversions.

| <b>From type</b>          | <b>To type</b>            |
|---------------------------|---------------------------|
| BIGINT (INT8)             | BOOLEAN                   |
|                           | CHAR                      |
|                           | DECIMAL (NUMERIC)         |
|                           | DOUBLE PRECISION (FLOAT8) |
|                           | INTEGER (INT, INT4)       |
|                           | REAL (FLOAT4)             |
|                           | SMALLINT (INT2)           |
|                           | VARCHAR                   |
| CHAR                      | VARCHAR                   |
| DATE                      | CHAR                      |
|                           | VARCHAR                   |
|                           | TIMESTAMP                 |
|                           | TIMESTAMPTZ               |
| DECIMAL (NUMERIC)         | BIGINT (INT8)             |
|                           | CHAR                      |
|                           | DOUBLE PRECISION (FLOAT8) |
|                           | INTEGER (INT, INT4)       |
|                           | REAL (FLOAT4)             |
|                           | SMALLINT (INT2)           |
|                           | VARCHAR                   |
| DOUBLE PRECISION (FLOAT8) | BIGINT (INT8)             |
|                           | CHAR                      |
|                           | DECIMAL (NUMERIC)         |
|                           | INTEGER (INT, INT4)       |
|                           | REAL (FLOAT4)             |
|                           | SMALLINT (INT2)           |
|                           | VARCHAR                   |
| INTEGER (INT, INT4)       | BIGINT (INT8)             |
|                           | BOOLEAN                   |
|                           | CHAR                      |
|                           | DECIMAL (NUMERIC)         |

| From type       | To type                   |
|-----------------|---------------------------|
|                 | DOUBLE PRECISION (FLOAT8) |
|                 | REAL (FLOAT4)             |
|                 | SMALLINT (INT2)           |
|                 | VARCHAR                   |
| REAL (FLOAT4)   | BIGINT (INT8)             |
|                 | CHAR                      |
|                 | DECIMAL (NUMERIC)         |
|                 | INTEGER (INT, INT4)       |
|                 | SMALLINT (INT2)           |
|                 | VARCHAR                   |
| SMALLINT (INT2) | BIGINT (INT8)             |
|                 | BOOLEAN                   |
|                 | CHAR                      |
|                 | DECIMAL (NUMERIC)         |
|                 | DOUBLE PRECISION (FLOAT8) |
|                 | INTEGER (INT, INT4)       |
|                 | REAL (FLOAT4)             |
|                 | VARCHAR                   |
| TIMESTAMP       | CHAR                      |
|                 | DATE                      |
|                 | VARCHAR                   |
|                 | TIMESTAMPTZ               |
|                 | TIME                      |
| TIMESTAMPTZ     | CHAR                      |
|                 | DATE                      |
|                 | VARCHAR                   |
|                 | TIMESTAMP                 |
|                 | TIMETZ                    |
| TIME            | VARCHAR                   |
|                 | TIMETZ                    |
| TIMETZ          | VARCHAR                   |

| From type | To type   |
|-----------|-----------|
|           | TIME      |
| GEOMETRY  | GEOGRAPHY |
| GEOGRAPHY | GEOMETRY  |

#### Note

Implicit conversions between TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ, or character strings use the current session time zone. For information about setting the current time zone, see [timezone \(p. 1700\)](#).

The GEOMETRY and GEOGRAPHY data types can't be implicitly converted to any other data type, except each other. For more information, see [CAST function \(p. 1066\)](#).

The VARBYTE data type can't be implicitly converted to any other data type. For more information, see [CAST function \(p. 1066\)](#).

## Using dynamic typing for the SUPER data type

Amazon Redshift uses dynamic typing to process schemaless SUPER data without the need to declare the data types before you use them in your query. Dynamic typing uses the results of navigating into SUPER data columns without having to explicitly cast them into Amazon Redshift types. For more information about using dynamic typing for SUPER data type, see [Dynamic typing \(p. 380\)](#).

You can cast SUPER values to and from other data types with some exceptions. For more information, see [Limitations \(p. 388\)](#).

## Collation sequences

Amazon Redshift doesn't support locale-specific or user-defined collation sequences. In general, the results of any predicate in any context could be affected by the lack of locale-specific rules for sorting and comparing data values. For example, ORDER BY expressions and functions such as MIN, MAX, and RANK return results based on binary UTF8 ordering of the data that does not take locale-specific characters into account.

# Expressions

## Topics

- [Simple expressions \(p. 601\)](#)
- [Compound expressions \(p. 602\)](#)
- [Expression lists \(p. 603\)](#)
- [Scalar subqueries \(p. 604\)](#)
- [Function expressions \(p. 604\)](#)

An expression is a combination of one or more values, operators, or functions that evaluate to a value. The data type of an expression is generally that of its components.

## Simple expressions

A simple expression is one of the following:

- A constant or literal value
- A column name or column reference
- A scalar function

- An aggregate (set) function
- A window function
- A scalar subquery

Examples of simple expressions include:

```
5+12
dateid
sales.qtysold * 100
sqrt (4)
max (qtysold)
(select max (qtysold) from sales)
```

## Compound expressions

A compound expression is a series of simple expressions joined by arithmetic operators. A simple expression used in a compound expression must return a numeric value.

### Syntax

```
expression
operator
expression | (compound_expression)
```

### Arguments

*expression*

A simple expression that evaluates to a value.

*operator*

A compound arithmetic expression can be constructed using the following operators, in this order of precedence:

- () : parentheses to control the order of evaluation
- + , - : positive and negative sign/operator
- ^ , |/ , ||/ : exponentiation, square root, cube root
- \* , / , % : multiplication, division, and modulo operators
- @ : absolute value
- + , - : addition and subtraction
- & , |, #, ~, <<, >> : AND, OR, NOT, shift left, shift right bitwise operators
- ||: concatenation

(*compound\_expression*)

Compound expressions can be nested using parentheses.

### Examples

Examples of compound expressions include the following.

```
('SMITH' || 'JONES')
sum(x) / y
sqrt(256) * avg(column)
rank() over (order by qtysold) / 100
```

```
(select (pricepaid - commission) from sales where dateid = 1882) * (qtysold)
```

Some functions can also be nested within other functions. For example, any scalar function can nest within another scalar function. The following example returns the sum of the absolute values of a set of numbers:

```
sum(abs(qtysold))
```

Window functions cannot be used as arguments for aggregate functions or other window functions. The following expression would return an error:

```
avg(rank() over (order by qtysold))
```

Window functions can have a nested aggregate function. The following expression sums sets of values and then ranks them:

```
rank() over (order by sum(qtysold))
```

## Expression lists

An expression list is a combination of expressions, and can appear in membership and comparison conditions (WHERE clauses) and in GROUP BY clauses.

### Syntax

```
expression , expression , ... | (expression, expression, ...)
```

### Arguments

*expression*

A simple expression that evaluates to a value. An expression list can contain one or more comma-separated expressions or one or more sets of comma-separated expressions. When there are multiple sets of expressions, each set must contain the same number of expressions, and be separated by parentheses. The number of expressions in each set must match the number of expressions before the operator in the condition.

### Examples

The following are examples of expression lists in conditions:

```
(1, 5, 10)
('THESE', 'ARE', 'STRINGS')
(('one', 'two', 'three'), ('blue', 'yellow', 'green'))
```

The number of expressions in each set must match the number in the first part of the statement:

```
select * from venue
where (venuecity, venuestate) in (('Miami', 'FL'), ('Tampa', 'FL'))
order by venueid;
```

| venueid | venuename               | venuecity | venuestate | venueseats |
|---------|-------------------------|-----------|------------|------------|
| 28      | American Airlines Arena | Miami     | FL         | 0          |

|                            |       |    |   |
|----------------------------|-------|----|---|
| 54   St. Pete Times Forum  | Tampa | FL | 0 |
| 91   Raymond James Stadium | Tampa | FL |   |
| (3 rows)                   |       |    |   |

## Scalar subqueries

A scalar subquery is a regular SELECT query in parentheses that returns exactly one value: one row with one column. The query is run and the returned value is used in the outer query. If the subquery returns zero rows, the value of the subquery expression is null. If it returns more than one row, Amazon Redshift returns an error. The subquery can refer to variables from the parent query, which will act as constants during any one invocation of the subquery.

You can use scalar subqueries in most statements that call for an expression. Scalar subqueries are not valid expressions in the following cases:

- As default values for expressions
- In GROUP BY and HAVING clauses

## Example

The following subquery computes the average price paid per sale across the entire year of 2008, then the outer query uses that value in the output to compare against the average price per sale per quarter:

```
select qtr, avg(pricepaid) as avg_saleprice_per_qtr,
(select avg(pricepaid)
from sales join date on sales.dateid=date.dateid
where year = 2008) as avg_saleprice_yearly
from sales join date on sales.dateid=date.dateid
where year = 2008
group by qtr
order by qtr;
qtr | avg_saleprice_per_qtr | avg_saleprice_yearly
-----+-----+-----
1 | 647.64 | 642.28
2 | 646.86 | 642.28
3 | 636.79 | 642.28
4 | 638.26 | 642.28
(4 rows)
```

## Function expressions

### Syntax

Any built-in can be used as an expression. The syntax for a function call is the name of a function followed by its argument list in parentheses.

|                                                    |
|----------------------------------------------------|
| <i>function ( [expression [, expression...]] )</i> |
|----------------------------------------------------|

### Arguments

*function*

Any built-in function.

*expression*

Any expression(s) matching the data type and parameter count expected by the function.

## Examples

```
abs (variable)
select avg (qtysold + 3) from sales;
select dateadd (day,30,caldate) as plus30days from date;
```

# Conditions

## Topics

- [Syntax \(p. 605\)](#)
- [Comparison condition \(p. 605\)](#)
- [Logical conditions \(p. 608\)](#)
- [Pattern-matching conditions \(p. 611\)](#)
- [BETWEEN range condition \(p. 619\)](#)
- [Null condition \(p. 621\)](#)
- [EXISTS condition \(p. 621\)](#)
- [IN condition \(p. 622\)](#)

A condition is a statement of one or more expressions and logical operators that evaluates to true, false, or unknown. Conditions are also sometimes referred to as predicates.

### Note

All string comparisons and LIKE pattern matches are case-sensitive. For example, 'A' and 'a' do not match. However, you can do a case-insensitive pattern match by using the ILIKE predicate.

## Syntax

```
comparison_condition
| logical_condition
| range_condition
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

## Comparison condition

Comparison conditions state logical relationships between two values. All comparison conditions are binary operators with a Boolean return type. Amazon Redshift supports the comparison operators described in the following table:

| Operator | Syntax           | Description                                  |
|----------|------------------|----------------------------------------------|
| <        | a < b            | Value a is less than value b.                |
| >        | a > b            | Value a is greater than value b.             |
| <=       | a <= b           | Value a is less than or equal to value b.    |
| >=       | a >= b           | Value a is greater than or equal to value b. |
| =        | a = b            | Value a is equal to value b.                 |
| <> or != | a <> b or a != b | Value a is not equal to value b.             |

| Operator                  | Syntax                         | Description                                                 |
|---------------------------|--------------------------------|-------------------------------------------------------------|
| ANY   SOME                | a = ANY(subquery)              | Value a is equal to any value returned by the subquery.     |
| ALL                       | a <> ALL or != ALL (subquery)) | Value a is not equal to any value returned by the subquery. |
| IS TRUE   FALSE   UNKNOWN | a IS TRUE                      | Value a is Boolean TRUE.                                    |

## Usage notes

= ANY | SOME

The ANY and SOME keywords are synonymous with the *IN* condition, and return true if the comparison is true for at least one value returned by a subquery that returns one or more values. Amazon Redshift supports only the = (equals) condition for ANY and SOME. Inequality conditions are not supported.

### Note

The ALL predicate is not supported.

<> ALL

The ALL keyword is synonymous with NOT IN (see [IN condition \(p. 622\)](#) condition) and returns true if the expression is not included in the results of the subquery. Amazon Redshift supports only the <> or != (not equals) condition for ALL. Other comparison conditions are not supported.

IS TRUE/FALSE/UNKNOWN

Non-zero values equate to TRUE, 0 equates to FALSE, and null equates to UNKNOWN. See the [Boolean type \(p. 592\)](#) data type.

## Examples

Here are some simple examples of comparison conditions:

```
a = 5
a < b
min(x) >= 5
qtysold = any (select qtysold from sales where dateid = 1882)
```

The following query returns venues with more than 10000 seats from the VENUE table:

```
select venueid, venuename, venueseats from venue
where venueseats > 10000
order by venueseats desc;
```

| venueid | venuename                      | venueseats |
|---------|--------------------------------|------------|
| 83      | FedExField                     | 91704      |
| 6       | New York Giants Stadium        | 80242      |
| 79      | Arrowhead Stadium              | 79451      |
| 78      | INVEESCO Field                 | 76125      |
| 69      | Dolphin Stadium                | 74916      |
| 67      | Ralph Wilson Stadium           | 73967      |
| 76      | Jacksonville Municipal Stadium | 73800      |

|                               |  |       |
|-------------------------------|--|-------|
| 89   Bank of America Stadium  |  | 73298 |
| 72   Cleveland Browns Stadium |  | 73200 |
| 86   Lambeau Field            |  | 72922 |
| ...                           |  |       |
| (57 rows)                     |  |       |

This example selects the users (USERID) from the USERS table who like rock music:

```
select userid from users where likerock = 't' order by 1 limit 5;

userid

3
5
6
13
16
(5 rows)
```

This example selects the users (USERID) from the USERS table where it is unknown whether they like rock music:

```
select firstname, lastname, likerock
from users
where likerock is unknown
order by userid limit 10;

firstname | lastname | likerock
-----+-----+-----
Rafael | Taylor |
Vladimir | Humphrey|
Barry | Roy |
Tamekah | Juarez |
Mufutau | Watkins |
Naida | Calderon|
Anika | Huff |
Bruce | Beck |
Mallory | Farrell |
Scarlett | Mayer |
(10 rows)
```

## Examples with a TIME column

The following example table TIME\_TEST has a column TIME\_VAL (type TIME) with three values inserted.

```
select time_val from time_test;

time_val

20:00:00
00:00:00.5550
00:58:00
```

The following example extracts the hours from each timetz\_val.

```
select time_val from time_test where time_val < '3:00';

time_val

00:00:00.5550
00:58:00
```

The following example compares two time literals.

```
select time '18:25:33.123456' = time '18:25:33.123456';
?column?

t
```

## Examples with a TIMETZ column

The following example table TIMETZ\_TEST has a column TIMETZ\_VAL (type TIMETZ) with three values inserted.

```
select timetz_val from timetz_test;
timetz_val

04:00:00+00
00:00:00.5550+00
05:58:00+00
```

The following example selects only the TIMETZ values less than 3:00:00 UTC. The comparison is made after converting the value to UTC.

```
select timetz_val from timetz_test where timetz_val < '3:00:00 UTC';
timetz_val

00:00:00.5550+00
```

The following example compares two TIMETZ literals. The time zone is ignored for the comparison.

```
select time '18:25:33.123456 PST' < time '19:25:33.123456 EST';
?column?

t
```

## Logical conditions

Logical conditions combine the result of two conditions to produce a single result. All logical conditions are binary operators with a Boolean return type.

### Syntax

```
expression
{ AND | OR }
expression
NOT expression
```

Logical conditions use a three-valued Boolean logic where the null value represents an unknown relationship. The following table describes the results for logical conditions, where E1 and E2 represent expressions:

| E1   | E2   | E1 AND E2 | E1 OR E2 | NOT E2 |
|------|------|-----------|----------|--------|
| TRUE | TRUE | TRUE      | TRUE     | FALSE  |

| E1      | E2      | E1 AND E2 | E1 OR E2 | NOT E2  |
|---------|---------|-----------|----------|---------|
| TRUE    | FALSE   | FALSE     | TRUE     | TRUE    |
| TRUE    | UNKNOWN | UNKNOWN   | TRUE     | UNKNOWN |
| FALSE   | TRUE    | FALSE     | TRUE     |         |
| FALSE   | FALSE   | FALSE     | FALSE    |         |
| FALSE   | UNKNOWN | FALSE     | UNKNOWN  |         |
| UNKNOWN | TRUE    | UNKNOWN   | TRUE     |         |
| UNKNOWN | FALSE   | FALSE     | UNKNOWN  |         |
| UNKNOWN | UNKNOWN | UNKNOWN   | UNKNOWN  |         |

The NOT operator is evaluated before AND, and the AND operator is evaluated before the OR operator. Any parentheses used may override this default order of evaluation.

### Examples

The following example returns USERID and USERNAME from the USERS table where the user likes both Las Vegas and sports:

```
select userid, username from users
where likevegas = 1 and likesports = 1
order by userid;

userid | username
-----+-----
1 | JSG99FHE
67 | TWU10MZT
87 | DUF19VXU
92 | HYP36WEQ
109 | FPL38HZK
120 | DMJ24GUZ
123 | QZR22XGQ
130 | ZQC82ALK
133 | LBN45WCH
144 | UCX04JKN
165 | TEY680EB
169 | AYQ83HG0
184 | TVX65AZX
...
(2128 rows)
```

The next example returns the USERID and USERNAME from the USERS table where the user likes Las Vegas, or sports, or both. This query returns all of the output from the previous example plus the users who like only Las Vegas or sports.

```
select userid, username from users
where likevegas = 1 or likesports = 1
order by userid;

userid | username
-----+-----
1 | JSG99FHE
2 | PGL08LJI
3 | IFT66TXU
```

```

5 | AEB55QTM
6 | NDQ15VBM
9 | MSD36KVR
10 | WKW41AIW
13 | QTF33MCG
15 | OWU78MTR
16 | ZMG93CDD
22 | RHT62AGI
27 | KOY02CVE
29 | HUH27PKK
...
(18968 rows)

```

The following query uses parentheses around the OR condition to find venues in New York or California where Macbeth was performed:

```

select distinct venue.name, venue.city
from venue join event on venue.venueid=event.venueid
where (venue.state = 'NY' or venue.state = 'CA') and eventname='Macbeth'
order by 2,1;

venue.name | venue.city
-----+-----
Geffen Playhouse | Los Angeles
Greek Theatre | Los Angeles
Royce Hall | Los Angeles
American Airlines Theatre | New York City
August Wilson Theatre | New York City
Belasco Theatre | New York City
Bernard B. Jacobs Theatre | New York City
...

```

Removing the parentheses in this example changes the logic and results of the query.

The following example uses the NOT operator:

```

select * from category
where not catid=1
order by 1;

catid | catgroup | catname | catdesc
-----+-----+-----+-----
2 | Sports | NHL | National Hockey League
3 | Sports | NFL | National Football League
4 | Sports | NBA | National Basketball Association
5 | Sports | MLS | Major League Soccer
...

```

The following example uses a NOT condition followed by an AND condition:

```

select * from category
where (not catid=1) and catgroup='Sports'
order by catid;

catid | catgroup | catname | catdesc
-----+-----+-----+-----
2 | Sports | NHL | National Hockey League
3 | Sports | NFL | National Football League
4 | Sports | NBA | National Basketball Association
5 | Sports | MLS | Major League Soccer
(4 rows)

```

## Pattern-matching conditions

### Topics

- [LIKE \(p. 611\)](#)
- [SIMILAR TO \(p. 614\)](#)
- [POSIX operators \(p. 616\)](#)

A pattern-matching operator searches a string for a pattern specified in the conditional expression and returns true or false depend on whether it finds a match. Amazon Redshift uses three methods for pattern matching:

- LIKE expressions

The LIKE operator compares a string expression, such as a column name, with a pattern that uses the wildcard characters % (percent) and \_ (underscore). LIKE pattern matching always covers the entire string. LIKE performs a case-sensitive match and ILIKE performs a case-insensitive match.

- SIMILAR TO regular expressions

The SIMILAR TO operator matches a string expression with a SQL standard regular expression pattern, which can include a set of pattern-matching metacharacters that includes the two supported by the LIKE operator. SIMILAR TO matches the entire string and performs a case-sensitive match.

- POSIX-style regular expressions

POSIX regular expressions provide a more powerful means for pattern matching than the LIKE and SIMILAR TO operators. POSIX regular expression patterns can match any portion of the string and performs a case-sensitive match.

Regular expression matching, using SIMILAR TO or POSIX operators, is computationally expensive. We recommend using LIKE whenever possible, especially when processing a very large number of rows. For example, the following queries are functionally identical, but the query that uses LIKE runs several times faster than the query that uses a regular expression:

```
select count(*) from event where eventname SIMILAR TO '%(Ring|Die)%';
select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die%';
```

### LIKE

The LIKE operator compares a string expression, such as a column name, with a pattern that uses the wildcard characters % (percent) and \_ (underscore). LIKE pattern matching always covers the entire string. To match a sequence anywhere within a string, the pattern must start and end with a percent sign.

LIKE is case-sensitive; ILIKE is case-insensitive.

### Syntax

```
expression [NOT] LIKE | ILIKE pattern [ESCAPE 'escape_char']
```

### Arguments

#### expression

A valid UTF-8 character expression, such as a column name.

## LIKE | ILIKE

LIKE performs a case-sensitive pattern match. ILIKE performs a case-insensitive pattern match for single-byte UTF-8 (ASCII) characters. To perform a case-insensitive pattern match for multibyte characters, use the [LOWER \(p. 1293\)](#) function on *expression* and *pattern* with a LIKE condition.

In contrast to comparison predicates, such as = and <>, LIKE and ILIKE predicates do not implicitly ignore trailing spaces. To ignore trailing spaces, use RTRIM or explicitly cast a CHAR column to VARCHAR.

*pattern*

A valid UTF-8 character expression with the pattern to be matched.

*escape\_char*

A character expression that will escape metacharacters characters in the pattern. The default is two backslashes ('\\').

If *pattern* does not contain metacharacters, then the pattern only represents the string itself; in that case LIKE acts the same as the equals operator.

Either of the character expressions can be CHAR or VARCHAR data types. If they differ, Amazon Redshift converts *pattern* to the data type of *expression*.

LIKE supports the following pattern-matching metacharacters:

| Operator | Description                                      |
|----------|--------------------------------------------------|
| %        | Matches any sequence of zero or more characters. |
| _        | Matches any single character.                    |

## Examples

The following table shows examples of pattern matching using LIKE:

| Expression        | Returns |
|-------------------|---------|
| 'abc' LIKE 'abc'  | True    |
| 'abc' LIKE 'a%    | True    |
| 'abc' LIKE '_B_'  | False   |
| 'abc' ILIKE '_B_' | True    |
| 'abc' LIKE 'c%'   | False   |

The following example finds all cities whose names start with "E":

```
select distinct city from users
where city like 'E%' order by city;
city

East Hartford
East Lansing
East Rutherford
```

```
East St. Louis
Easthampton
Easton
Eatontown
Eau Claire
...
```

The following example finds users whose last name contains "ten":

```
select distinct lastname from users
where lastname like '%ten%' order by lastname;
lastname

Christensen
Wooten
...
```

The following example finds cities whose third and fourth characters are "ea". The command uses ILIKE to demonstrate case insensitivity:

```
select distinct city from users where city ilike '__EA%' order by city;
city

Brea
Clearwater
Great Falls
Ocean City
Olean
Wheaton
(6 rows)
```

The following example uses the default escape string (\) to search for strings that include "start\_" (the text start followed by an underscore \_):

```
select tablename, "column" from pg_table_def
where "column" like '%start_%'
limit 5;

tablename | column
-----+-----
stl_s3client | start_time
stl_tr_conflict | xact_start_ts
stl undone | undo_start_ts
stl unload_log | start_time
stl vacuum_detail | start_row
(5 rows)
```

The following example specifies '^' as the escape character, then uses the escape character to search for strings that include "start\_" (the text start followed by an underscore \_):

```
select tablename, "column" from pg_table_def
where "column" like '%start^_%' escape '^'
limit 5;

tablename | column
-----+-----
stl_s3client | start_time
stl_tr_conflict | xact_start_ts
stl undone | undo_start_ts
stl unload_log | start_time
stl vacuum_detail | start_row
```

(5 rows)

## SIMILAR TO

The SIMILAR TO operator matches a string expression, such as a column name, with a SQL standard regular expression pattern. A SQL regular expression pattern can include a set of pattern-matching metacharacters, including the two supported by the [LIKE \(p. 611\)](#) operator.

The SIMILAR TO operator returns true only if its pattern matches the entire string, unlike POSIX regular expression behavior, where the pattern can match any portion of the string.

SIMILAR TO performs a case-sensitive match.

### Note

Regular expression matching using SIMILAR TO is computationally expensive. We recommend using LIKE whenever possible, especially when processing a very large number of rows. For example, the following queries are functionally identical, but the query that uses LIKE runs several times faster than the query that uses a regular expression:

```
select count(*) from event where eventname SIMILAR TO '%(Ring|Die)%';
select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die%';
```

## Syntax

```
expression [NOT] SIMILAR TO pattern [ESCAPE 'escape_char']
```

## Arguments

### *expression*

A valid UTF-8 character expression, such as a column name.

### SIMILAR TO

SIMILAR TO performs a case-sensitive pattern match for the entire string in *expression*.

### *pattern*

A valid UTF-8 character expression representing a SQL standard regular expression pattern.

### *escape\_char*

A character expression that will escape metacharacters in the pattern. The default is two backslashes ('\\').

If *pattern* does not contain metacharacters, then the pattern only represents the string itself.

Either of the character expressions can be CHAR or VARCHAR data types. If they differ, Amazon Redshift converts *pattern* to the data type of *expression*.

SIMILAR TO supports the following pattern-matching metacharacters:

| Operator | Description                                       |
|----------|---------------------------------------------------|
| %        | Matches any sequence of zero or more characters.  |
| -        | Matches any single character.                     |
|          | Denotes alternation (either of two alternatives). |

| Operator | Description                                                                             |
|----------|-----------------------------------------------------------------------------------------|
| *        | Repeat the previous item zero or more times.                                            |
| +        | Repeat the previous item one or more times.                                             |
| ?        | Repeat the previous item zero or one time.                                              |
| {m}      | Repeat the previous item exactly <i>m</i> times.                                        |
| {m,}     | Repeat the previous item <i>m</i> or more times.                                        |
| {m, n}   | Repeat the previous item at least <i>m</i> and not more than <i>n</i> times.            |
| ( )      | Parentheses group items into a single logical item.                                     |
| [...]    | A bracket expression specifies a character class, just as in POSIX regular expressions. |

## Examples

The following table shows examples of pattern matching using SIMILAR TO:

| Expression                                                     | Returns |
|----------------------------------------------------------------|---------|
| 'abc' SIMILAR TO 'abc'                                         | True    |
| 'abc' SIMILAR TO '_b_'                                         | True    |
| 'abc' SIMILAR TO '_A_'                                         | False   |
| 'abc' SIMILAR TO '%(b d)%'                                     | True    |
| 'abc' SIMILAR TO '(b c)%'                                      | False   |
| 'AbcAbcdefgefg12efgefg12' SIMILAR TO '((Ab)?c)+d((efg)+(12))+' | True    |
| 'aaaaaab11111xy' SIMILAR TO 'a{6}_[0-9]{5}(x y){2}'            | True    |
| '\$.87' SIMILAR TO '\$[0-9]+(. [0-9][0-9])?'                   | True    |

The following example finds cities whose names contain "E" or "H":

```
SELECT DISTINCT city FROM users
WHERE city SIMILAR TO '%E%|%H%' ORDER BY city LIMIT 5;

city

Agoura Hills
Auburn Hills
Benton Harbor
Beverly Hills
Chicago Heights
```

The following example uses the default escape string ('\\') to search for strings that include "\_":

```
SELECT tablename, "column" FROM pg_table_def
WHERE "column" SIMILAR TO '%start_%'
ORDER BY tablename, "column" LIMIT 5;

 tablename | column
-----+-----
stcs_abort_idle | idle_start_time
stcs_abort_idle | txn_start_time
stcs_analyze_compression | start_time
stcs_auto_worker_levels | start_level
stcs_auto_worker_levels | start_wlm_occupancy
```

The following example specifies '^' as the escape string, then uses the escape string to search for strings that include "\_":

```
SELECT tablename, "column" FROM pg_table_def
WHERE "column" SIMILAR TO '%start^_%' ESCAPE '^'
ORDER BY tablename, "column" LIMIT 5;

 tablename | column
-----+-----
stcs_abort_idle | idle_start_time
stcs_abort_idle | txn_start_time
stcs_analyze_compression | start_time
stcs_auto_worker_levels | start_level
stcs_auto_worker_levels | start_wlm_occupancy
```

## POSIX operators

POSIX regular expressions provide a more powerful means for pattern matching than the [LIKE \(p. 611\)](#) and [SIMILAR TO \(p. 614\)](#) operators. POSIX regular expression patterns can match any portion of a string, unlike the SIMILAR TO operator, which returns true only if its pattern matches the entire string.

### Note

Regular expression matching using POSIX operators is computationally expensive. We recommend using LIKE whenever possible, especially when processing a very large number of rows. For example, the following queries are functionally identical, but the query that uses LIKE runs several times faster than the query that uses a regular expression:

```
select count(*) from event where eventname ~ '.*\b(Ring|Die)\b.*';
select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die%';
```

## Syntax

```
expression [!] ~ pattern
```

## Arguments

*expression*

A valid UTF-8 character expression, such as a column name.

!

Negation operator.

~

Perform a case-sensitive match for any substring of *expression*.

*pattern*

A string literal that represents a SQL standard regular expression pattern.

If *pattern* does not contain wildcard characters, then the pattern only represents the string itself.

To search for strings that include metacharacters, such as '.', '\*' | '?', and so on, escape the character using two backslashes ('\\'). Unlike SIMILAR TO and LIKE, POSIX regular expression syntax does not support a user-defined escape character.

Either of the character expressions can be CHAR or VARCHAR data types. If they differ, Amazon Redshift converts *pattern* to the data type of *expression*.

All of the character expressions can be CHAR or VARCHAR data types. If the expressions differ in data type, Amazon Redshift converts them to the data type of *expression*.

POSIX pattern matching supports the following metacharacters:

| POSIX  | Description                                                                                                                                                                                              |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| .      | Matches any single character.                                                                                                                                                                            |
| *      | Matches zero or more occurrences.                                                                                                                                                                        |
| +      | Matches one or more occurrences.                                                                                                                                                                         |
| ?      | Matches zero or one occurrence.                                                                                                                                                                          |
|        | Specifies alternative matches; for example, E   H means E or H.                                                                                                                                          |
| ^      | Matches the beginning-of-line character.                                                                                                                                                                 |
| \$     | Matches the end-of-line character.                                                                                                                                                                       |
| \$     | Matches the end of the string.                                                                                                                                                                           |
| []     | Brackets specify a matching list, that should match one expression in the list. A caret (^) precedes a nonmatching list, which matches any character except for the expressions represented in the list. |
| ( )    | Parentheses group items into a single logical item.                                                                                                                                                      |
| {m}    | Repeat the previous item exactly <i>m</i> times.                                                                                                                                                         |
| {m,}   | Repeat the previous item <i>m</i> or more times.                                                                                                                                                         |
| {m, n} | Repeat the previous item at least <i>m</i> and not more than <i>n</i> times.                                                                                                                             |
| [: :]  | Matches any character within a POSIX character class. In the following character classes, Amazon Redshift supports only ASCII characters:<br>[:alnum:], [:alpha:], [:lower:], [:upper:]                  |

Amazon Redshift supports the following POSIX character classes.

| Character Class | Description                       |
|-----------------|-----------------------------------|
| [:alnum:]       | All ASCII alphanumeric characters |

| Character Class | Description                               |
|-----------------|-------------------------------------------|
| [:alpha:]       | All ASCII alphabetic characters           |
| [:blank:]       | All blank space characters                |
| [:cntrl:]       | All control characters (nonprinting)      |
| [:digit:]       | All numeric digits                        |
| [:lower:]       | All lowercase ASCII alphabetic characters |
| [:punct:]       | All punctuation characters                |
| [:space:]       | All space characters (nonprinting)        |
| [:upper:]       | All uppercase ASCII alphabetic characters |
| [:xdigit:]      | All valid hexadecimal characters          |

Amazon Redshift supports the following Perl-influenced operators in regular expressions. Escape the operator using two backslashes ('\\').

| Operator | Description                 | Equivalent character class expression |
|----------|-----------------------------|---------------------------------------|
| \\d      | A digit character           | [:digit:]                             |
| \\D      | A nondigit character        | [^[:digit:]]                          |
| \\w      | A word character            | [:word:]                              |
| \\W      | A nonword character         | [^[:word:]]                           |
| \\s      | A white space character     | [:space:]                             |
| \\S      | A non–white space character | [^[:space:]]                          |

## Examples

The following table shows examples of pattern matching using POSIX operators:

| Expression                                            | Returns |
|-------------------------------------------------------|---------|
| 'abc' ~ 'abc'                                         | True    |
| 'abc' ~ 'a'                                           | True    |
| 'abc' ~ 'A'                                           | False   |
| 'abc' ~ '.*(b d).*'                                   | True    |
| 'abc' ~ '(b c).*'                                     | True    |
| 'AbcAbcdefgefg12efgefg12' ~ '((Ab)?c)+d((efg)+(12))+' | True    |
| 'aaaaaab11111xy' ~ 'a{6}.[1]{5}(x y){2}'              | True    |

| Expression                                | Returns |
|-------------------------------------------|---------|
| '\$0.87' ~ '\\$\[0-9]+(\.\[0-9]\[0-9]\)?' | True    |
| 'ab c' ~ '[:space:]'                      | True    |
| 'ab c' ~ '\s'                             | True    |
| ' ' ~ '\S'                                | False   |

The following example finds cities whose names contain E or H:

```
SELECT DISTINCT city FROM users
WHERE city ~ '.*E.*|.*H.*' ORDER BY city LIMIT 5;

 city

Agoura Hills
Auburn Hills
Benton Harbor
Beverly Hills
Chicago Heights
```

The following example uses the escape string ('\') to search for strings that include a period.

```
SELECT venuename FROM venue
WHERE venuename ~ '.*\\..*'
ORDER BY venueid;

 venuename

St. Pete Times Forum
Jobing.com Arena
Hubert H. Humphrey Metrodome
U.S. Cellular Field
Superpages.com Center
E.J. Nutter Center
Bernard B. Jacobs Theatre
St. James Theatre
```

## BETWEEN range condition

A BETWEEN condition tests expressions for inclusion in a range of values, using the keywords BETWEEN and AND.

### Syntax

```
expression [NOT] BETWEEN expression AND expression
```

Expressions can be numeric, character, or datetime data types, but they must be compatible. The range is inclusive.

### Examples

The first example counts how many transactions registered sales of either 2, 3, or 4 tickets:

```
select count(*) from sales
```

```
where qtysold between 2 and 4;

count

104021
(1 row)
```

The range condition includes the begin and end values.

```
select min(dateid), max(dateid) from sales
where dateid between 1900 and 1910;

min | max
-----+-----
1900 | 1910
```

The first expression in a range condition must be the lesser value and the second expression the greater value. The following example will always return zero rows due to the values of the expressions:

```
select count(*) from sales
where qtysold between 4 and 2;

count

0
(1 row)
```

However, applying the NOT modifier will invert the logic and produce a count of all rows:

```
select count(*) from sales
where qtysold not between 4 and 2;

count

172456
(1 row)
```

The following query returns a list of venues with 20000 to 50000 seats:

```
select venueid, venuename, venueseats from venue
where venueseats between 20000 and 50000
order by venueseats desc;

venueid | venuename | venueseats
-----+-----+
116 | Busch Stadium | 49660
106 | Rangers BallPark in Arlington | 49115
96 | Oriole Park at Camden Yards | 48876
...
(22 rows)
```

The following example demonstrates using BETWEEN for date values:

```
select salesid, qtysold, pricepaid, commission, saletime
from sales
where eventid between 1000 and 2000
 and saletime between '2008-01-01' and '2008-01-03'
order by saletime asc;
```

| salesid | qtysold | pricepaid | commission | saletime       |
|---------|---------|-----------|------------|----------------|
| 65082   | 4       | 472       | 70.8       | 1/1/2008 06:06 |
| 110917  | 1       | 337       | 50.55      | 1/1/2008 07:05 |
| 112103  | 1       | 241       | 36.15      | 1/2/2008 03:15 |
| 137882  | 3       | 1473      | 220.95     | 1/2/2008 05:18 |
| 40331   | 2       | 58        | 8.7        | 1/2/2008 05:57 |
| 110918  | 3       | 1011      | 151.65     | 1/2/2008 07:17 |
| 96274   | 1       | 104       | 15.6       | 1/2/2008 07:18 |
| 150499  | 3       | 135       | 20.25      | 1/2/2008 07:20 |
| 68413   | 2       | 158       | 23.7       | 1/2/2008 08:12 |

Note that although BETWEEN's range is inclusive, dates default to having a time value of 00:00:00. The only valid January 3 row for the sample query would be a row with a saletime of 1/3/2008 00:00:00.

## Null condition

The null condition tests for nulls, when a value is missing or unknown.

### Syntax

```
expression IS [NOT] NULL
```

### Arguments

*expression*

Any expression such as a column.

IS NULL

Is true when the expression's value is null and false when it has a value.

IS NOT NULL

Is false when the expression's value is null and true when it has a value.

### Example

This example indicates how many times the SALES table contains null in the QTYSOLD field:

```
select count(*) from sales
where qtysold is null;
count

0
(1 row)
```

## EXISTS condition

EXISTS conditions test for the existence of rows in a subquery, and return true if a subquery returns at least one row. If NOT is specified, the condition returns true if a subquery returns no rows.

### Syntax

```
[NOT] EXISTS (table_subquery)
```

## Arguments

**EXISTS**

Is true when the *table\_subquery* returns at least one row.

**NOT EXISTS**

Is true when the *table\_subquery* returns no rows.

*table\_subquery*

A subquery that evaluates to a table with one or more columns and one or more rows.

## Example

This example returns all date identifiers, one time each, for each date that had a sale of any kind:

```
select dateid from date
where exists (
 select 1 from sales
 where date.dateid = sales.dateid
)
order by dateid;

dateid

1827
1828
1829
...
```

## IN condition

An IN condition tests a value for membership in a set of values or in a subquery.

### Syntax

```
expression [NOT] IN (expr_list | table_subquery)
```

## Arguments

*expression*

A numeric, character, or datetime expression that is evaluated against the *expr\_list* or *table\_subquery* and must be compatible with the data type of that list or subquery.

*expr\_list*

One or more comma-delimited expressions, or one or more sets of comma-delimited expressions bounded by parentheses.

*table\_subquery*

A subquery that evaluates to a table with one or more rows, but is limited to only one column in its select list.

**IN | NOT IN**

IN returns true if the expression is a member of the expression list or query. NOT IN returns true if the expression is not a member. IN and NOT IN return NULL and no rows are returned in the

following cases: If *expression* yields null; or if there are no matching *expr\_list* or *table\_subquery* values and at least one of these comparison rows yields null.

## Examples

The following conditions are true only for those values listed:

```
qtysold in (2, 4, 5)
date.day in ('Mon', 'Tues')
date.month not in ('Oct', 'Nov', 'Dec')
```

## Optimization for Large IN Lists

To optimize query performance, an IN list that includes more than 10 values is internally evaluated as a scalar array. IN lists with fewer than 10 values are evaluated as a series of OR predicates. This optimization is supported for SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP, and TIMESTAMPTZ data types.

Look at the EXPLAIN output for the query to see the effect of this optimization. For example:

```
explain select * from sales
QUERY PLAN

XN Seq Scan on sales (cost=0.00..6035.96 rows=86228 width=53)
Filter: (salesid = ANY ('{1,2,3,4,5,6,7,8,9,10,11}'::integer[]))
(2 rows)
```

# SQL commands

The SQL language consists of commands that you use to create and manipulate database objects, run queries, load tables, and modify the data in tables.

Amazon Redshift is based on PostgreSQL. Amazon Redshift and PostgreSQL have a number of important differences that you must be aware of as you design and develop your data warehouse applications. For more information about how Amazon Redshift SQL differs from PostgreSQL, see [Amazon Redshift and PostgreSQL \(p. 565\)](#).

### Note

The maximum size for a single SQL statement is 16 MB.

### Topics

- [ABORT \(p. 626\)](#)
- [ALTER DATABASE \(p. 627\)](#)
- [ALTER DATASHARE \(p. 629\)](#)
- [ALTER DEFAULT PRIVILEGES \(p. 631\)](#)
- [ALTER GROUP \(p. 633\)](#)
- [ALTER IDENTITY PROVIDER \(p. 634\)](#)
- [ALTER MASKING POLICY \(p. 635\)](#)
- [ALTER MATERIALIZED VIEW \(p. 636\)](#)
- [ALTER ROLE \(p. 636\)](#)
- [ALTER PROCEDURE \(p. 637\)](#)
- [ALTER SCHEMA \(p. 638\)](#)

- [ALTER TABLE \(p. 640\)](#)
- [ALTER TABLE APPEND \(p. 656\)](#)
- [ALTER USER \(p. 659\)](#)
- [ANALYZE \(p. 663\)](#)
- [ANALYZE COMPRESSION \(p. 665\)](#)
- [ATTACH MASKING POLICY \(p. 667\)](#)
- [ATTACH RLS POLICY \(p. 668\)](#)
- [BEGIN \(p. 669\)](#)
- [CALL \(p. 670\)](#)
- [CANCEL \(p. 672\)](#)
- [CLOSE \(p. 674\)](#)
- [COMMENT \(p. 674\)](#)
- [COMMIT \(p. 676\)](#)
- [COPY \(p. 677\)](#)
- [CREATE DATABASE \(p. 745\)](#)
- [CREATE DATASHARE \(p. 754\)](#)
- [CREATE EXTERNAL FUNCTION \(p. 755\)](#)
- [CREATE EXTERNAL SCHEMA \(p. 762\)](#)
- [CREATE EXTERNAL TABLE \(p. 768\)](#)
- [CREATE FUNCTION \(p. 788\)](#)
- [CREATE GROUP \(p. 792\)](#)
- [CREATE IDENTITY PROVIDER \(p. 792\)](#)
- [CREATE LIBRARY \(p. 793\)](#)
- [CREATE MASKING POLICY \(p. 796\)](#)
- [CREATE MATERIALIZED VIEW \(p. 796\)](#)
- [CREATE MODEL \(p. 800\)](#)
- [CREATE PROCEDURE \(p. 815\)](#)
- [CREATE RLS POLICY \(p. 819\)](#)
- [CREATE ROLE \(p. 820\)](#)
- [CREATE SCHEMA \(p. 821\)](#)
- [CREATE TABLE \(p. 823\)](#)
- [CREATE TABLE AS \(p. 838\)](#)
- [CREATE USER \(p. 846\)](#)
- [CREATE VIEW \(p. 851\)](#)
- [DEALLOCATE \(p. 854\)](#)
- [DECLARE \(p. 855\)](#)
- [DELETE \(p. 858\)](#)
- [DESC DATASHARE \(p. 860\)](#)
- [DESC IDENTITY PROVIDER \(p. 861\)](#)
- [DETACH MASKING POLICY \(p. 861\)](#)
- [DETACH RLS POLICY \(p. 862\)](#)
- [DROP DATABASE \(p. 863\)](#)
- [DROP DATASHARE \(p. 864\)](#)
- [DROP FUNCTION \(p. 865\)](#)

- [DROP GROUP \(p. 866\)](#)
- [DROP IDENTITY PROVIDER \(p. 867\)](#)
- [DROP LIBRARY \(p. 867\)](#)
- [DROP MASKING POLICY \(p. 868\)](#)
- [DROP MODEL \(p. 868\)](#)
- [DROP MATERIALIZED VIEW \(p. 869\)](#)
- [DROP PROCEDURE \(p. 870\)](#)
- [DROP RLS POLICY \(p. 870\)](#)
- [DROP ROLE \(p. 871\)](#)
- [DROP SCHEMA \(p. 872\)](#)
- [DROP TABLE \(p. 874\)](#)
- [DROP USER \(p. 876\)](#)
- [DROP VIEW \(p. 878\)](#)
- [END \(p. 879\)](#)
- [EXECUTE \(p. 880\)](#)
- [EXPLAIN \(p. 881\)](#)
- [FETCH \(p. 886\)](#)
- [GRANT \(p. 887\)](#)
- [INSERT \(p. 902\)](#)
- [INSERT \(external table\) \(p. 906\)](#)
- [LOCK \(p. 908\)](#)
- [MERGE \(p. 909\)](#)
- [PREPARE \(p. 912\)](#)
- [REFRESH MATERIALIZED VIEW \(p. 914\)](#)
- [RESET \(p. 915\)](#)
- [REVOKE \(p. 916\)](#)
- [ROLLBACK \(p. 927\)](#)
- [SELECT \(p. 928\)](#)
- [SELECT INTO \(p. 976\)](#)
- [SET \(p. 977\)](#)
- [SET SESSION AUTHORIZATION \(p. 980\)](#)
- [SET SESSION CHARACTERISTICS \(p. 981\)](#)
- [SHOW \(p. 981\)](#)
- [SHOW EXTERNAL TABLE \(p. 982\)](#)
- [SHOW DATABASES \(p. 984\)](#)
- [SHOW MODEL \(p. 984\)](#)
- [SHOW DATASHARES \(p. 986\)](#)
- [SHOW PROCEDURE \(p. 987\)](#)
- [SHOW TABLE \(p. 988\)](#)
- [SHOW VIEW \(p. 989\)](#)
- [START TRANSACTION \(p. 990\)](#)
- [TRUNCATE \(p. 990\)](#)
- [UNLOAD \(p. 991\)](#)
- [UPDATE \(p. 1010\)](#)

- [VACUUM \(p. 1015\)](#)

## ABORT

Stops the currently running transaction and discards all updates made by that transaction. ABORT has no effect on already completed transactions.

This command performs the same function as the ROLLBACK command. For information, see [ROLLBACK \(p. 927\)](#).

### Syntax

```
ABORT [WORK | TRANSACTION]
```

### Parameters

WORK

Optional keyword.

TRANSACTION

Optional keyword; WORK and TRANSACTION are synonyms.

### Example

The following example creates a table then starts a transaction where data is inserted into the table. The ABORT command then rolls back the data insertion to leave the table empty.

The following command creates an example table called MOVIE\_GROSS:

```
create table movie_gross(name varchar(30), gross bigint);
```

The next set of commands starts a transaction that inserts two data rows into the table:

```
begin;
insert into movie_gross values ('Raiders of the Lost Ark', 23400000);
insert into movie_gross values ('Star Wars', 10000000);
```

Next, the following command selects the data from the table to show that it was successfully inserted:

```
select * from movie_gross;
```

The command output shows that both rows are successfully inserted:

| name                    | gross    |
|-------------------------|----------|
| Raiders of the Lost Ark | 23400000 |
| Star Wars               | 10000000 |
| (2 rows)                |          |

This command now rolls back the data changes to where the transaction began:

```
abort;
```

Selecting data from the table now shows an empty table:

```
select * from movie_gross;

name | gross
-----+-----
(0 rows)
```

## ALTER DATABASE

Changes the attributes of a database.

### Syntax

```
ALTER DATABASE database_name
[RENAME TO new_name
| OWNER TO new_owner
| CONNECTION LIMIT { limit | UNLIMITED }
| COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }]
[ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }]
```

### Parameters

*database\_name*

Name of the database to alter. Typically, you alter a database that you are not currently connected to; in any case, the changes take effect only in subsequent sessions. You can change the owner of the current database, but you can't rename it:

```
alter database tickit rename to newtickit;
ERROR: current database may not be renamed
```

RENAME TO

Renames the specified database. For more information about valid names, see [Names and identifiers \(p. 571\)](#). You can't rename the dev, padb\_harvest, template0, or template1 databases, and you can't rename the current database. Only the database owner or a [superuser \(p. 537\)](#) can rename a database; non-superuser owners must also have the CREATEDB privilege.

*new\_name*

New database name.

OWNER TO

Changes the owner of the specified database. You can change the owner of the current database or some other database. Only a superuser can change the owner.

*new\_owner*

New database owner. The new owner must be an existing database user with write privileges. For more information about user privileges, see [GRANT \(p. 887\)](#).

CONNECTION LIMIT { *limit* | UNLIMITED }

The maximum number of database connections users are permitted to have open concurrently. The limit is not enforced for superusers. Use the UNLIMITED keyword to permit the maximum number

of concurrent connections. A limit on the number of connections for each user might also apply. For more information, see [CREATE USER \(p. 846\)](#). The default is UNLIMITED. To view current connections, query the [STV\\_SESSIONS \(p. 1597\)](#) system view.

**Note**

If both user and database connection limits apply, an unused connection slot must be available that is within both limits when a user attempts to connect.

**COLLATE { CASE\_SENSITIVE | CASE\_INSENSITIVE }**

A clause that specifies whether string search or comparison is case-sensitive or case-insensitive.

You can change the case sensitivity of the current database which is empty.

You must have the privilege to the current database to change case sensitivity. Superusers or database owners with the CREATE DATABASE privilege can also change database case sensitivity.

**ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }**

A clause that specifies the isolation level used when queries run against a database.

- SERIALIZABLE isolation – provides full serializability for concurrent transactions. For more information, see [Serializable isolation \(p. 119\)](#).
- SNAPSHOT isolation – provides an isolation level with protection against update and delete conflicts.

For more information about isolation levels, see [CREATE DATABASE \(p. 745\)](#).

Consider the following items when altering the isolation level of a database:

- You must have the superuser or CREATE DATABASE privilege to the current database to change the database isolation level.
- You can't alter the isolation level of the dev database.
- You can't alter the isolation level within a transaction block.
- The alter isolation level command fails if other users are connected to the database.
- The alter isolation level command can alter the isolation level settings of the current session.

## Usage notes

ALTER DATABASE commands apply to subsequent sessions not current sessions. You must reconnect to the altered database to see the effect of the change.

## Examples

The following example renames a database named TICKIT\_SANDBOX to TICKIT\_TEST:

```
alter database tickit_sandbox rename to tickit_test;
```

The following example changes the owner of the TICKIT database (the current database) to DWUSER:

```
alter database tickit owner to dwuser;
```

The following example changes the database case sensitivity of the sampledb database:

```
ALTER DATABASE sampledb COLLATE CASE_INSENSITIVE;
```

The following example alters a database named **samp1edb** with SNAPSHOT isolation level.

```
ALTER DATABASE sampledb ISOLATION LEVEL SNAPSHOT;
```

## ALTER DATASHARE

Changes the definition of a datashare. You can add objects or remove objects using ALTER DATASHARE. You can only change a datashare in the current database. Add or remove objects from the associated database to a datashare. The owner of the datashare with the required permissions on the datashare objects to be added or removed can alter the datashare.

### Required privileges

Following are required privileges for ALTER DATASHARE:

- Superuser.
- User with the ALTER DATASHARE privilege.
- Users who have the ALTER or ALL privilege on the datashare.
- To add specific objects to a datashare, users must have the privilege on the objects. For this case, users should be the owners of objects or have SELECT, USAGE, or ALL privileges on the objects.

### Syntax

The following syntax illustrates how to add or remove objects to the datashare.

```
ALTER DATASHARE datashare_name { ADD | REMOVE } {
 TABLE schema.table [, ...]
 | SCHEMA schema [, ...]
 | FUNCTION schema.sql_udf (argtype,...) [, ...]
 | ALL TABLES IN SCHEMA schema [, ...]
 | ALL FUNCTIONS IN SCHEMA schema [, ...] }
```

The following syntax illustrates how to configure the properties of the datashare.

```
ALTER DATASHARE datashare_name {
 [SET PUBLICACCESSIBLE [=] TRUE | FALSE]
 [SET INCLUDEDNEW [=] TRUE | FALSE FOR SCHEMA schema] }
```

### Parameters

*datashare\_name*

The name of the datashare to be altered.

**ADD | REMOVE**

A clause that specifies whether to add objects to or remove objects from the datashare.

**TABLE *schema.table* [, ...]**

The name of the table or view in the specified schema to add to the datashare.

**SCHEMA *schema* [, ...]**

The name of the schema to add to the datashare.

**FUNCTION *schema.sql\_udf* (*argtype*,...) [, ...]**

The name of the user-defined SQL function with argument types to add to the datashare.

ALL TABLES IN SCHEMA *schema* [, ...]

A clause that specifies whether to add all tables and views in the specified schema to the datashare.

ALL FUNCTIONS IN SCHEMA *schema* [, ...]}

A clause that specifies adding all functions in the specified schema to the datashare.

[ SET PUBLICACCESSIBLE [=] TRUE | FALSE ]

A clause that specifies whether a datashare can be shared to clusters that are publicly accessible.

[ SET INCLUDENEW [=] TRUE | FALSE FOR SCHEMA *schema* ]

A clause that specifies whether to add any future tables, views, or SQL user-defined functions (UDFs) created in the specified schema to the datashare. Current tables, views, or SQL UDFs in the specified schema aren't added to the datashare. Only superusers can change this property for each datashare-schema pair. By default, the INCLUDENEW clause is false.

## ALTER DATASHARE usage notes

- The following users can alter a datashare:
  - A superuser
  - The owner of the datashare
  - Users that have ALTER or ALL privilege on the datashare
- To add specific objects to a datashare, users must have the correct privileges on the objects. Users should be the owners of objects or have SELECT, USAGE, or ALL privileges on the objects.
- You can share schemas, tables, regular views, late-binding views, materialized views, and SQL user-defined functions (UDFs). Add a schema to a datashare first before adding objects in the schema.

When you add a schema, Amazon Redshift doesn't add all the objects under it. You must add them explicitly.

- We recommend that you create AWS Data Exchange datashares with the publicly accessible setting turned on.
- In general, we recommend that you don't alter an AWS Data Exchange datashare to turn off public accessibility using the ALTER DATASHARE statement. If you do, the AWS accounts that have access to the datashare lose access if their clusters are publicly accessible. Performing this type of alteration can breach data product terms in AWS Data Exchange. For an exception to this recommendation, see following.

The following example shows an error when an AWS Data Exchange datashare is created with the setting turned off.

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
ERROR: Alter of ADX-managed datashare salesshare requires session variable
datashare_break_glass_session_var to be set to value 'c670ba4db22f4b'
```

To allow altering an AWS Data Exchange datashare to turn off the publicly accessible setting, set the following variable and run the ALTER DATASHARE statement again.

```
SET datashare_break_glass_session_var to 'c670ba4db22f4b';
```

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
```

In this case, Amazon Redshift generates a random one-time value to set the session variable to allow ALTER DATASHARE SET PUBLICACCESSIBLE FALSE for an AWS Data Exchange datashare.

## Examples

The following example adds the public.ticket\_sales\_redshift table to the datashare salesshare.

```
ALTER DATASHARE salesshare ADD TABLE public.ticket_sales_redshift;
```

The following example adds all tables to the datashare salesshare.

```
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

The following example removes the public.ticket\_sales\_redshift table from the datashare salesshare.

```
ALTER DATASHARE salesshare REMOVE TABLE public.ticket_sales_redshift;
```

## ALTER DEFAULT PRIVILEGES

Defines the default set of access privileges to be applied to objects that are created in the future by the specified user. By default, users can change only their own default access privileges. Only a superuser can specify default privileges for other users.

You can apply default privileges to roles, users, or user groups. You can set default privileges globally for all objects created in the current database, or for objects created only in the specified schemas.

Default privileges apply only to new objects. Running ALTER DEFAULT PRIVILEGES doesn't change privileges on existing objects.

To view information about the default privileges for database users, query the [PG\\_DEFAULT\\_ACL \(p. 1673\)](#) system catalog table.

For more information about privileges, see [GRANT \(p. 887\)](#).

## Required privileges

Following are required privileges for ALTER DEFAULT PRIVILEGES:

- Superuser
- Users with the ALTER DEFAULT PRIVILEGES privilege
- Users changing their own default access privileges
- Users setting privileges for schemas that they have access privileges to

## Syntax

```
ALTER DEFAULT PRIVILEGES
[FOR USER target_user [, ...]]
[IN SCHEMA schema_name [, ...]]
grant_or_revoke_clause

where grant_or_revoke_clause is one of:

GRANT { { SELECT | INSERT | UPDATE | DELETE | DROP | REFERENCES } [,...] | ALL
[PRIVILEGES] }
```

```

ON TABLES
TO { user_name [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC }
[, ...]

GRANT { EXECUTE | ALL [PRIVILEGES] }
ON FUNCTIONS
TO { user_name [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC }
[, ...]

GRANT { EXECUTE | ALL [PRIVILEGES] }
ON PROCEDURES
TO { user_name [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC }
[, ...]

REVOKE [GRANT OPTION FOR] { { SELECT | INSERT | UPDATE | DELETE | REFERENCES } [, ...] |
ALL [PRIVILEGES] }
ON TABLES
FROM user_name [, ...] [RESTRICT]

REVOKE { { SELECT | INSERT | UPDATE | DELETE | REFERENCES } [, ...] | ALL [PRIVILEGES] }
ON TABLES
FROM { ROLE role_name | GROUP group_name | PUBLIC } [, ...] [RESTRICT]

REVOKE [GRANT OPTION FOR] { EXECUTE | ALL [PRIVILEGES] }
ON FUNCTIONS
FROM user_name [, ...] [RESTRICT]

REVOKE { EXECUTE | ALL [PRIVILEGES] }
ON FUNCTIONS
FROM { ROLE role_name | GROUP group_name | PUBLIC } [, ...] [RESTRICT]

REVOKE [GRANT OPTION FOR] { EXECUTE | ALL [PRIVILEGES] }
ON PROCEDURES
FROM user_name [, ...] [RESTRICT]

REVOKE { EXECUTE | ALL [PRIVILEGES] }
ON PROCEDURES
FROM { ROLE role_name | GROUP group_name | PUBLIC } [, ...] [RESTRICT]

```

## Parameters

**FOR USER *target\_user***

Optional. The name of the user for which default privileges are defined. Only a superuser can specify default privileges for other users. The default value is the current user.

**IN SCHEMA *schema\_name***

Optional. If an IN SCHEMA clause appears, the specified default privileges are applied to new objects created in the specified *schema\_name*. In this case, the user or user group that is the target of ALTER DEFAULT PRIVILEGES must have CREATE privilege for the specified schema. Default privileges that are specific to a schema are added to existing global default privileges. By default, default privileges are applied globally to the entire database.

**GRANT**

The set of privileges to grant to the specified users or groups for all new tables, functions, or stored procedures created by the specified user. You can set the same privileges and options with the GRANT clause that you can with the [GRANT \(p. 887\)](#) command.

**WITH GRANT OPTION**

A clause that indicates that the user receiving the privileges can in turn grant the same privileges to others. You can't grant WITH GRANT OPTION to a group or to PUBLIC.

**TO user\_name | ROLE role\_name | GROUP group\_name**

The name of the user, role, or user group to which the specified default privileges are applied.

**REVOKE**

The set of privileges to revoke from the specified users or groups for all new tables, functions, or stored procedures created by the specified user. You can set the same privileges and options with the REVOKE clause that you can with the [REVOKE \(p. 916\)](#) command.

**GRANT OPTION FOR**

A clause that revokes only the option to grant a specified privilege to other users and doesn't revoke the privilege itself. You can't revoke GRANT OPTION from a group or from PUBLIC.

**FROM user\_name | ROLE role\_name | GROUP group\_name**

The name of the user, role, or user group from which the specified privileges are revoked by default.

**RESTRICT**

The RESTRICT option revokes only those privileges that the user directly granted. This is the default.

## Examples

Suppose that you want to allow any user in the user group `report_readers` to view all tables created by the user `report_admin`. In this case, run the following command as a superuser.

```
alter default privileges for user report_admin grant select on tables to group report_readers;
```

In the following example, the first command grants SELECT privileges on all new tables you create.

```
alter default privileges grant select on tables to public;
```

The following example grants INSERT privilege to the `sales_admin` user group for all new tables and views that you create in the `sales` schema.

```
alter default privileges in schema sales grant insert on tables to group sales_admin;
```

The following example reverses the ALTER DEFAULT PRIVILEGES command in the preceding example.

```
alter default privileges in schema sales revoke insert on tables from group sales_admin;
```

By default, the PUBLIC user group has execute permission for all new user-defined functions. To revoke public execute permissions for your new functions and then grant execute permission only to the `dev_test` user group, run the following commands.

```
alter default privileges revoke execute on functions from public;
alter default privileges grant execute on functions to group dev_test;
```

## ALTER GROUP

Changes a user group. Use this command to add users to the group, drop users from the group, or rename the group.

## Syntax

```
ALTER GROUP group_name
{
 ADD USER username [, ...] |
 DROP USER username [, ...] |
 RENAME TO new_name
}
```

## Parameters

*group\_name*

Name of the user group to modify.

ADD

Adds a user to a user group.

DROP

Removes a user from a user group.

*username*

Name of the user to add to the group or drop from the group.

RENAME TO

Renames the user group. Group names beginning with two underscores are reserved for Amazon Redshift internal use. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

*new\_name*

New name of the user group.

## Examples

The following example adds a user named DWUSER to the ADMIN\_GROUP group:

```
alter group admin_group
add user dwuser;
```

The following example renames the group ADMIN\_GROUP to ADMINISTRATORS:

```
alter group admin_group
rename to administrators;
```

## ALTER IDENTITY PROVIDER

Alters an identity provider to assign new parameters and values. When you run this command, all previously set parameter values are deleted before the new values are assigned. Only a superuser can alter an identity provider.

## Syntax

```
ALTER IDENTITY PROVIDER identity_provider_name
```

PARAMETERS *parameter\_string*

## Parameters

*identity\_provider\_name*

Name of the new identity provider. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

*parameter\_string*

A string containing a properly formatted JSON object that contains parameters and values required for the specific identity provider.

## Example

The following example alters an identity provider named *oauth\_standard*.

```
ALTER IDENTITY PROVIDER oauth_standard
PARAMETERS '{"issuer":"https://sts.windows.net/2sdfdsf-d475-420d-b5ac-667adad7c702/",
"client_id":"87f4aa26-78b7-410e-bf29-57b39929ef9a",
"client_secret":"BUAH~ewrqewrqwerUUY^%tHe1oNZShoiU7",
"audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift"]
}'
```

## ALTER MASKING POLICY

Alters an existing dynamic data masking policy. For more information on dynamic data masking, see [Dynamic data masking \(p. 555\)](#).

Superusers and users or roles that have the sys:secadmin role can alter a masking policy.

## Syntax

```
ALTER MASKING POLICY policy_name
USING (masking_expression);
```

## Parameters

*policy\_name*

The name of the masking policy. This must be the name of a masking policy that already exists in the database.

*masking\_expression*

The SQL expression used to transform the target columns. It can be written using data manipulation functions such as String manipulation functions, or in conjunction with user-defined functions written in SQL, Python, or with AWS Lambda.

The expression must match the original expression's input columns and data types. For example, if the original masking policy's input columns were `sample_1` FLOAT and `sample_2` VARCHAR(10), you wouldn't be able to alter the masking policy to take a third column, or make the policy take a FLOAT and a BOOLEAN. If you use a constant as your masking expression, you must explicitly cast it to a type that matches the input type.

You must have the USAGE permission on any user-defined functions that you use in the masking expression.

## ALTER MATERIALIZED VIEW

Enables automatic refreshing of a materialized view.

### Syntax

```
ALTER MATERIALIZED VIEW mv_name
[AUTO REFRESH { YES | NO }]
[ROW LEVEL SECURITY { ON | OFF } [FOR DATASHARES]];
```

### Parameters

AUTO REFRESH { YES | NO }

A clause that turns on or off automatic refreshing of a materialized view. For more information about automatic refresh of materialized views, see [Refreshing a materialized view \(p. 202\)](#).

ROW LEVEL SECURITY { ON | OFF }

A clause that turns on or off row-level security for a materialized view. When row-level security is turned on, you can read only the rows that the row-level policy permits for access. When there isn't any policy granting a user access to the table or view, the user can't see any records. For more information, see [CREATE RLS POLICY \(p. 819\)](#).

FOR DATASHARES

A clause that determines whether a materialized view is RLS-protected over datashares. By default, a RLS-protected materialized view is also RLS-protected over datashares.

### Examples

The following example enables the `tickets_mv` materialized view to be automatically refreshed.

```
ALTER MATERIALIZED VIEW tickets_mv AUTO REFRESH YES
```

## ALTER ROLE

Renames a role or changes the owner. For a list of Amazon Redshift system-defined roles, see [the section called "Amazon Redshift system-defined roles" \(p. 542\)](#).

### Required permissions

Following are the required permissions for ALTER ROLE:

- Superuser
- Users with the ALTER ROLE permissions

### Syntax

```
ALTER ROLE role [WITH]
```

```
{ { RENAME TO role } | { OWNER TO user_name } }[, ...]
[EXTERNALID TO external_id]
```

## Parameters

*role*

The name of the role to be altered.

RENAME TO

A new name for the role.

OWNER TO *user\_name*

A new owner for the role.

EXTERNALID TO *external\_id*

A new external ID for the role, which is associated with an identity provider. For more information, see [Native identity provider \(IdP\) federation for Amazon Redshift](#).

## Examples

The following example changes the name of a role from `sample_role1` to `sample_role2`.

```
ALTER ROLE sample_role1 WITH RENAME TO sample_role2;
```

The following example changes the owner of the role.

```
ALTER ROLE sample_role1 WITH OWNER TO user1
```

The syntax of the `ALTER ROLE` is similar to `ALTER PROCEDURE` following.

```
ALTER PROCEDURE first_quarter_revenue(bigint, numeric) RENAME TO quarterly_revenue;
```

The following example changes the owner of a procedure to `etl_user`.

```
ALTER PROCEDURE quarterly_revenue(bigint, numeric) OWNER TO etl_user;
```

The following example updates a role `sample_role1` with a new external ID that is associated with an identity provider.

```
ALTER ROLE sample_role1 EXTERNALID TO "XYZ456";
```

## ALTER PROCEDURE

Renames a procedure or changes the owner. Both the procedure name and data types, or signature, are required. Only the owner or a superuser can rename a procedure. Only a superuser can change the owner of a procedure.

## Syntax

```
ALTER PROCEDURE sp_name [([[argname] [argmode] argtype [, ...]])]
```

```
RENAME TO new_name
```

```
ALTER PROCEDURE sp_name [([[argname] [argmode] argtype [, ...]])]
OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
```

## Parameters

*sp\_name*

The name of the procedure to be altered. Either specify just the name of the procedure in the current search path, or use the format `schema_name.sp_procedure_name` to use a specific schema.

*[argname] [argmode] argtype*

A list of argument names, argument modes, and data types. Only the input data types are required, which are used to identify the stored procedure. Alternatively, you can provide the full signature used to create the procedure including the input and output parameters with their modes.

*new\_name*

A new name for the stored procedure.

*new\_owner* | CURRENT\_USER | SESSION\_USER

A new owner for the stored procedure.

## Examples

The following example changes the name of a procedure from `first_quarter_revenue` to `quarterly_revenue`.

```
ALTER PROCEDURE first_quarter_revenue(volume INOUT bigint, at_price IN numeric,
result OUT int) RENAME TO quarterly_revenue;
```

This example is equivalent to the following.

```
ALTER PROCEDURE first_quarter_revenue(bigint, numeric) RENAME TO quarterly_revenue;
```

The following example changes the owner of a procedure to `etl_user`.

```
ALTER PROCEDURE quarterly_revenue(bigint, numeric) OWNER TO etl_user;
```

## ALTER SCHEMA

Changes the definition of an existing schema. Use this command to rename a schema or change the owner of a schema. For example, rename an existing schema to preserve a backup copy of that schema when you plan to create a new version of that schema. For more information about schemas, see [CREATE SCHEMA \(p. 821\)](#).

To view the configured schema quotas, see [SVV\\_SCHEMA\\_QUOTA\\_STATE \(p. 1454\)](#).

To view the records where schema quotas were exceeded, see [STL\\_SCHEMA\\_QUOTA\\_VIOLATIONS \(p. 1557\)](#).

## Required privileges

Following are required privileges for ALTER SCHEMA:

- Superuser
- Owner of the schema

When you change a schema name, note that objects using the old name, such as stored procedures or materialized views, must be updated to use the new name.

## Syntax

```
ALTER SCHEMA schema_name
{
 RENAME TO new_name |
 OWNER TO new_owner |
 QUOTA { quota [MB | GB | TB] | UNLIMITED }
}
```

## Parameters

*schema\_name*

The name of the database schema to be altered.

RENAME TO

A clause that renames the schema.

*new\_name*

The new name of the schema. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

OWNER TO

A clause that changes the owner of the schema.

*new\_owner*

The new owner of the schema.

QUOTA

The maximum amount of disk space that the specified schema can use. This space is the collective size of all tables under the specified schema. Amazon Redshift converts the selected value to megabytes. Gigabytes is the default unit of measurement when you don't specify a value.

For more information about configuring schema quotas, see [CREATE SCHEMA \(p. 821\)](#).

## Examples

The following example renames the SALES schema to US\_SALES.

```
alter schema sales
rename to us_sales;
```

The following example gives ownership of the US\_SALES schema to the user DWUSER.

```
alter schema us_sales
owner to dwuser;
```

The following example changes the quota to 300 GB and removes the quota.

```
alter schema us_sales QUOTA 300 GB;
alter schema us_sales QUOTA UNLIMITED;
```

## ALTER TABLE

This command changes the definition of a Amazon Redshift table or Amazon Redshift Spectrum external table. This command updates the values and properties set by [CREATE TABLE \(p. 823\)](#) or [CREATE EXTERNAL TABLE \(p. 768\)](#).

You can't run ALTER TABLE on an external table within a transaction block (BEGIN ... END). For more information about transactions, see [Serializable isolation \(p. 119\)](#).

ALTER TABLE locks the table for read and write operations until the transaction enclosing the ALTER TABLE operation completes, unless it's specifically stated in the documentation that you can query data or perform other operations on the table while it is altered.

### Required privileges

The user that alters a table needs the proper privilege for the command to succeed. Depending on the ALTER TABLE command, one of the following privileges is required.

- Superuser
- Users with the ALTER TABLE privilege
- Table owner with the USAGE privilege on the schema

### Syntax

```
ALTER TABLE table_name
{
 ADD table_constraint
 | DROP CONSTRAINT constraint_name [RESTRICT | CASCADE]
 | OWNER TO new_owner
 | RENAME TO new_name
 | RENAME COLUMN column_name TO new_name
 | ALTER COLUMN column_name TYPE updated_varchar_data_type_size
 | ALTER COLUMN column_name ENCODE new_encode_type
 | ALTER COLUMN column_name ENCODE encode_type,
 | ALTER COLUMN column_name ENCODE encode_type,;
 | ALTER DISTKEY column_name
 | ALTER DISTSTYLE ALL
 | ALTER DISTSTYLE EVEN
 | ALTER DISTSTYLE KEY DISTKEY column_name
 | ALTER DISTSTYLE AUTO
 | ALTER [COMPOUND] SORTKEY (column_name [,...])
 | ALTER SORTKEY AUTO
 | ALTER SORTKEY NONE
 | ALTER ENCODE AUTO
 | ADD [COLUMN] column_name column_type
 [DEFAULT default_expr]
 [ENCODE encoding]
 [NOT NULL | NULL]
```

```
[COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }] |
| DROP [COLUMN] column_name [RESTRICT | CASCADE]
| ROW LEVEL SECURITY { ON | OFF } [FOR DATASHARES]}
```

where *table\_constraint* is:

```
[CONSTRAINT constraint_name]
{ UNIQUE (column_name [, ...])
| PRIMARY KEY (column_name [, ...])
| FOREIGN KEY (column_name [, ...])
 REFERENCES reftable [(refcolumn)]}
```

The following options apply only to external tables:

```
SET LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file' }
| SET FILE FORMAT format
| SET TABLE PROPERTIES ('property_name'='property_value')
| PARTITION (partition_column=partition_value [, ...])
 SET LOCATION { 's3://bucket/folder' | 's3://bucket/manifest_file' }
| ADD [IF NOT EXISTS]
 PARTITION (partition_column=partition_value [, ...]) LOCATION { 's3://bucket/folder'
 | 's3://bucket/manifest_file' }
 [, ...]
| DROP PARTITION (partition_column=partition_value [, ...])
```

To reduce the time to run the ALTER TABLE command, you can combine some clauses of the ALTER TABLE command.

Amazon Redshift supports the following combinations of the ALTER TABLE clauses:

```
ALTER TABLE tablename ALTER SORTKEY (column_list), ALTER DISTKEY column_Id;
ALTER TABLE tablename ALTER DISTKEY column_Id, ALTER SORTKEY (column_list);
ALTER TABLE tablename ALTER SORTKEY (column_list), ALTER DISTSTYLE ALL;
ALTER TABLE tablename ALTER DISTSTYLE ALL, ALTER SORTKEY (column_list);
```

## Parameters

### *table\_name*

The name of the table to alter. Either specify just the name of the table, or use the format *schema\_name.table\_name* to use a specific schema. External tables must be qualified by an external schema name. You can also specify a view name if you are using the ALTER TABLE statement to rename a view or change its owner. The maximum length for the table name is 127 bytes; longer names are truncated to 127 bytes. You can use UTF-8 multibyte characters up to a maximum of four bytes. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

### ADD *table\_constraint*

A clause that adds the specified constraint to the table. For descriptions of valid *table\_constraint* values, see [CREATE TABLE \(p. 823\)](#).

#### Note

You can't add a primary-key constraint to a nullable column. If the column was originally created with the NOT NULL constraint, you can add the primary-key constraint.

### DROP CONSTRAINT *constraint\_name*

A clause that drops the named constraint from the table. To drop a constraint, specify the constraint name, not the constraint type. To view table constraint names, run the following query.

```
select constraint_name, constraint_type
```

```
from information_schema.table_constraints;
```

#### RESTRICT

A clause that removes only the specified constraint. RESTRICT is an option for DROP CONSTRAINT. RESTRICT can't be used with CASCADE.

#### CASCADE

A clause that removes the specified constraint and anything dependent on that constraint. CASCADE is an option for DROP CONSTRAINT. CASCADE can't be used with RESTRICT.

#### OWNER TO *new\_owner*

A clause that changes the owner of the table (or view) to the *new\_owner* value.

#### RENAME TO *new\_name*

A clause that renames a table (or view) to the value specified in *new\_name*. The maximum table name length is 127 bytes; longer names are truncated to 127 bytes.

You can't rename a permanent table to a name that begins with '#'. A table name beginning with '#' indicates a temporary table.

You can't rename an external table.

#### ALTER COLUMN *column\_name* TYPE *updated\_varchar\_data\_type\_size*

A clause that changes the size of a column defined as a VARCHAR data type. This clause only supports altering the size of a VARCHAR data type. Consider the following limitations:

- You can't alter a column with compression encodings BYTEDICT, RUNLENGTH, TEXT255, or TEXT32K.
- You can't decrease the size less than maximum size of existing data.
- You can't alter columns with default values.
- You can't alter columns with UNIQUE, PRIMARY KEY, or FOREIGN KEY.
- You can't alter columns within a transaction block (BEGIN ... END). For more information about transactions, see [Serializable isolation \(p. 119\)](#).

#### ALTER COLUMN *column\_name* ENCODE *new\_encode\_type*

A clause that changes the compression encoding of a column. If you specify compression encoding for a column, the table is no longer set to ENCODE AUTO. For information on compression encoding, see [Working with column compression \(p. 38\)](#).

When you change compression encoding for a column, the table remains available to query.

Consider the following limitations:

- You can't alter a column to the same encoding as currently defined for the column.
- You can't alter the encoding for a column in a table with an interleaved sortkey.

#### ALTER COLUMN *column\_name* ENCODE *encode\_type*, ALTER COLUMN *column\_name* ENCODE *encode\_type*, ....;

A clause that changes the compression encoding of multiple columns in a single command. For information on compression encoding, see [Working with column compression \(p. 38\)](#).

When you change compression encoding for a column, the table remains available to query.

Consider the following limitations:

- You can't alter a column to the same or different encoding type multiple times in a single command.

- You can't alter a column to the same encoding as currently defined for the column.
- You can't alter the encoding for a column in a table with an interleaved sortkey.

#### ALTER DISTSTYLE ALL

A clause that changes the existing distribution style of a table to ALL. Consider the following:

- An ALTER DISTSTYLE, ALTER SORTKEY, and VACUUM can't run concurrently on the same table.
  - If VACUUM is currently running, then running ALTER DISTSTYLE ALL returns an error.
  - If ALTER DISTSTYLE ALL is running, then a background vacuum doesn't start on a table.
- The ALTER DISTSTYLE ALL command is not supported for tables with interleaved sort keys and temporary tables.
- If the distribution style was previously defined as AUTO, then the table is no longer a candidate for automatic table optimization.

For more information about DISTSTYLE ALL, see [CREATE TABLE \(p. 823\)](#).

#### ALTER DISTSTYLE EVEN

A clause that changes the existing distribution style of a table to EVEN. Consider the following:

- An ALTER DISTSYTLE, ALTER SORTKEY, and VACUUM can't run concurrently on the same table.
  - If VACUUM is currently running, then running ALTER DISTSTYLE EVEN returns an error.
  - If ALTER DISTSTYLE EVEN is running, then a background vacuum doesn't start on a table.
- The ALTER DISTSTYLE EVEN command is not supported for tables with interleaved sort keys and temporary tables.
- If the distribution style was previously defined as AUTO, then the table is no longer a candidate for automatic table optimization.

For more information about DISTSTYLE EVEN, see [CREATE TABLE \(p. 823\)](#).

#### ALTER DISTKEY *column\_name* or ALTER DISTSTYLE KEY DISTKEY *column\_name*

A clause that changes the column used as the distribution key of a table. Consider the following:

- VACUUM and ALTER DISTKEY can't run concurrently on the same table.
  - If VACUUM is already running, then ALTER DISTKEY returns an error.
  - If ALTER DISTKEY is running, then background vacuum doesn't start on a table.
  - If ALTER DISTKEY is running, then foreground vacuum returns an error.
- You can only run one ALTER DISTKEY command on a table at a time.
- The ALTER DISTKEY command is not supported for tables with interleaved sort keys.
- If the distribution style was previously defined as AUTO, then the table is no longer a candidate for automatic table optimization.

When specifying DISTSTYLE KEY, the data is distributed by the values in the DISTKEY column. For more information about DISTSTYLE, see [CREATE TABLE \(p. 823\)](#).

#### ALTER DISTSTYLE AUTO

A clause that changes the existing distribution style of a table to AUTO.

When you alter a distribution style to AUTO, the distribution style of the table is set to the following:

- A small table with DISTSTYLE ALL is converted to AUTO(ALL).
- A small table with DISTSTYLE EVEN is converted to AUTO(ALL).
- A small table with DISTSTYLE KEY is converted to AUTO(ALL).
- A large table with DISTSTYLE ALL is converted to AUTO(EVEN).
- A large table with DISTSTYLE EVEN is converted to AUTO(EVEN).

- A large table with DISTSTYLE KEY is converted to AUTO(KEY) and the DISTKEY is preserved. In this case, Amazon Redshift makes no changes to the table.

If Amazon Redshift determines that a new distribution style or key will improve the performance of queries, then Amazon Redshift might change the distribution style or key of your table in the future. For example, Amazon Redshift might convert a table with a DISTSTYLE of AUTO(KEY) to AUTO(EVEN), or vice versa. For more information about behavior when distribution keys are altered, including data redistribution and locks, see [Amazon Redshift Advisor recommendations](#).

For more information about DISTSTYLE AUTO, see [CREATE TABLE \(p. 823\)](#).

To view the distribution style of a table, query the SVV\_TABLE\_INFO system catalog view. For more information, see [SVV\\_TABLE\\_INFO \(p. 1457\)](#). To view the Amazon Redshift Advisor recommendations for tables, query the SVV\_ALTER\_TABLE\_RECOMMENDATIONS system catalog view. For more information, see [SVV\\_ALTER\\_TABLE\\_RECOMMENDATIONS \(p. 1415\)](#). To view the actions taken by Amazon Redshift, query the SVL\_AUTO\_WORKER\_ACTION system catalog view. For more information, see [SVL\\_AUTO\\_WORKER\\_ACTION \(p. 1629\)](#).

#### ALTER [COMPOUND] SORTKEY ( *column\_name* [...] )

A clause that changes or adds the sort key used for a table.

When you alter a sort key, the compression encoding of columns in the new or original sort key can change. If no encoding is explicitly defined for the table, then Amazon Redshift automatically assigns compression encodings as follows:

- Columns that are defined as sort keys are assigned RAW compression.
- Columns that are defined as BOOLEAN, REAL, or DOUBLE PRECISION data types are assigned RAW compression.
- Columns that are defined as SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP, or Timestamptz are assigned AZ64 compression.
- Columns that are defined as CHAR or VARCHAR are assigned LZO compression.

Consider the following:

- You can define a maximum of 400 columns for a sort key per table.
- You can alter an interleaved sort key to a compound sort key or no sort key. However, you can't alter a compound sort key to an interleaved sort key.
- If the sort key was previously defined as AUTO, then the table is no longer a candidate for automatic table optimization.
- Amazon Redshift recommends using RAW encoding (no compression) for columns defined as sort keys. When you alter a column to choose it as a sort key, the column's compression is changed to RAW compression (no compression). This can increase the amount of storage required by the table. How much the table size increases depend on the specific table definition and table contents. For more information about compression, see [Compression encodings \(p. 39\)](#)

When data is loaded into a table, the data is loaded in the order of the sort key. When you alter the sort key, Amazon Redshift reorders the data. For more information about SORTKEY, see [CREATE TABLE \(p. 823\)](#).

#### ALTER SORTKEY AUTO

A clause that changes or adds the sort key of the target table to AUTO.

When you alter a sort key to AUTO, Amazon Redshift preserves the existing sort key of the table.

If Amazon Redshift determines that a new sort key will improve the performance of queries, then Amazon Redshift might change the sort key of your table in the future.

For more information about SORTKEY AUTO, see [CREATE TABLE \(p. 823\)](#).

To view the sort key of a table, query the SVV\_TABLE\_INFO system catalog view. For more information, see [SVV\\_TABLE\\_INFO \(p. 1457\)](#). To view the Amazon Redshift Advisor recommendations for tables, query the SVV\_ALTER\_TABLE\_RECOMMENDATIONS system catalog view. For more information, see [SVV\\_ALTER\\_TABLE\\_RECOMMENDATIONS \(p. 1415\)](#). To view the actions taken by Amazon Redshift, query the SVL\_AUTO\_WORKER\_ACTION system catalog view. For more information, see [SVL\\_AUTO\\_WORKER\\_ACTION \(p. 1629\)](#).

#### ALTER SORTKEY NONE

A clause that removes the sort key of the target table.

If the sort key was previously defined as AUTO, then the table is no longer a candidate for automatic table optimization.

#### ALTER ENCODE AUTO

A clause that changes the encoding type of the target table columns to AUTO. When you alter encoding to AUTO, Amazon Redshift preserves the existing encoding type of the columns in the table. Then, if Amazon Redshift determines that a new encoding type can improve query performance, Amazon Redshift can change the encoding type of the table columns.

If you alter one or more columns to specify an encoding, Amazon Redshift no longer automatically adjusts encoding for all columns in the table. The columns retain the current encode settings.

The following actions don't affect the ENCODE AUTO setting for the table:

- Renaming the table.
- Altering the DISTSTYLE or SORTKEY setting for the table.
- Adding or dropping a column with an ENCODE setting.
- Using the COMPUPDATE option of the COPY command. For more information, see [Data load operations \(p. 711\)](#).

To view the encoding of a table, query the SVV\_TABLE\_INFO system catalog view. For more information, see [SVV\\_TABLE\\_INFO \(p. 1457\)](#).

#### RENAME COLUMN *column\_name* TO *new\_name*

A clause that renames a column to the value specified in *new\_name*. The maximum column name length is 127 bytes; longer names are truncated to 127 bytes. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

#### ADD [ COLUMN ] *column\_name*

A clause that adds a column with the specified name to the table. You can add only one column in each ALTER TABLE statement.

You can't add a column that is the distribution key (DISTKEY) or a sort key (SORTKEY) of the table.

You can't use an ALTER TABLE ADD COLUMN command to modify the following table and column attributes:

- UNIQUE
- PRIMARY KEY
- REFERENCES (foreign key)
- IDENTITY or GENERATED BY DEFAULT AS IDENTITY

The maximum column name length is 127 bytes; longer names are truncated to 127 bytes. The maximum number of columns you can define in a single table is 1,600.

The following restrictions apply when adding a column to an external table:

- You can't add a column to an external table with the column constraints DEFAULT, ENCODE, NOT NULL, or NULL.
- You can't add columns to an external table that's defined using the AVRO file format.
- If pseudocolumns are enabled, the maximum number of columns that you can define in a single external table is 1,598. If pseudocolumns aren't enabled, the maximum number of columns that you can define in a single table is 1,600.

For more information, see [CREATE EXTERNAL TABLE \(p. 768\)](#).

*column\_type*

The data type of the column being added. For CHAR and VARCHAR columns, you can use the MAX keyword instead of declaring a maximum length. MAX sets the maximum length to 4,096 bytes for CHAR or 65,535 bytes for VARCHAR. The maximum size of a GEOMETRY object is 1,048,447 bytes.

For information about the data types that Amazon Redshift supports, see [Data types \(p. 574\)](#).

*DEFAULT default\_expr*

A clause that assigns a default data value for the column. The data type of *default\_expr* must match the data type of the column. The DEFAULT value must be a variable-free expression. Subqueries, cross-references to other columns in the current table, and user-defined functions aren't allowed.

The *default\_expr* is used in any INSERT operation that doesn't specify a value for the column. If no default value is specified, the default value for the column is null.

If a COPY operation encounters a null field on a column that has a DEFAULT value and a NOT NULL constraint, the COPY command inserts the value of the *default\_expr*.

DEFAULT isn't supported for external tables.

*ENCODE encoding*

The compression encoding for a column. By default, Amazon Redshift automatically manages compression encoding for all columns in a table if you don't specify compression encoding for any column in the table or if you specify the ENCODE AUTO option for the table.

If you specify compression encoding for any column in the table or if you don't specify the ENCODE AUTO option for the table, Amazon Redshift automatically assigns compression encoding to columns for which you don't specify compression encoding as follows:

- All columns in temporary tables are assigned RAW compression by default.
- Columns that are defined as sort keys are assigned RAW compression.
- Columns that are defined as BOOLEAN, REAL, DOUBLE PRECISION, GEOMETRY, or GEOGRAPHY data type are assigned RAW compression.
- Columns that are defined as SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP, or Timestamptz are assigned AZ64 compression.
- Columns that are defined as CHAR, VARCHAR, or VARBYTE are assigned LZO compression.

**Note**

If you don't want a column to be compressed, explicitly specify RAW encoding.

The following [compression encodings \(p. 39\)](#) are supported:

- AZ64
- BYTEDICT
- DELTA
- DELTA32K
- LZO

- MOSTLY8
- MOSTLY16
- MOSTLY32
- RAW (no compression)
- RUNLENGTH
- TEXT255
- TEXT32K
- ZSTD

ENCODE isn't supported for external tables.

NOT NULL | NULL

NOT NULL specifies that the column isn't allowed to contain null values. NULL, the default, specifies that the column accepts null values.

NOT NULL and NULL aren't supported for external tables.

COLLATE { CASE\_SENSITIVE | CASE\_INSENSITIVE }

A clause that specifies whether string search or comparison on the column is CASE\_SENSITIVE or CASE\_INSENSITIVE. The default value is the same as the current case sensitivity configuration of the database.

To find the database collation information, use the following command:

```
select db_collation();
```

```
db_collation

case_sensitive
(1 row)
```

DROP [ COLUMN ] *column\_name*

The name of the column to delete from the table.

You can't drop the last column in a table. A table must have at least one column.

You can't drop a column that is the distribution key (DISTKEY) or a sort key (SORTKEY) of the table. The default behavior for DROP COLUMN is RESTRICT if the column has any dependent objects, such as a view, primary key, foreign key, or UNIQUE restriction.

The following restrictions apply when dropping a column from an external table:

- You can't drop a column from an external table if the column is used as a partition.
- You can't drop a column from an external table that is defined using the AVRO file format.
- RESTRICT and CASCADE are ignored for external tables.
- You can't drop the columns of the policy table referenced inside the policy definition unless you drop or detach the policy. This also applies when the CASCADE option is specified. You can drop other columns in the policy table.

For more information, see [CREATE EXTERNAL TABLE \(p. 768\)](#).

RESTRICT

When used with DROP COLUMN, RESTRICT means that column to be dropped isn't dropped, in these cases:

- If a defined view references the column that is being dropped
- If a foreign key references the column
- If the column takes part in a multipart key

RESTRICT can't be used with CASCADE.

RESTRICT and CASCADE are ignored for external tables.

#### CASCADE

When used with DROP COLUMN, removes the specified column and anything dependent on that column. CASCADE can't be used with RESTRICT.

RESTRICT and CASCADE are ignored for external tables.

The following options apply only to external tables.

**SET LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest\_file' }**

The path to the Amazon S3 folder that contains the data files or a manifest file that contains a list of Amazon S3 object paths. The buckets must be in the same AWS Region as the Amazon Redshift cluster. For a list of supported AWS Regions, see [Amazon Redshift Spectrum considerations \(p. 249\)](#). For more information about using a manifest file, see LOCATION in the CREATE EXTERNAL TABLE Parameters (p. 770) reference.

**SET FILE FORMAT *format***

The file format for external data files.

Valid formats are as follows:

- AVRO
- PARQUET
- RCFILE
- SEQUENCEFILE
- TEXTFILE

**SET TABLE PROPERTIES ( '*property\_name*'='*property\_value*' )**

A clause that sets the table definition for table properties for an external table.

#### Note

Table properties are case-sensitive.

**'numRows'='row\_count'**

A property that sets the numRows value for the table definition. To explicitly update an external table's statistics, set the numRows property to indicate the size of the table. Amazon Redshift doesn't analyze external tables to generate the table statistics that the query optimizer uses to generate a query plan. If table statistics aren't set for an external table, Amazon Redshift generates a query execution plan. This plan is based on an assumption that external tables are the larger tables and local tables are the smaller tables.

**'skip.header.line.count'='line\_count'**

A property that sets number of rows to skip at the beginning of each source file.

**PARTITION ( *partition\_column*=*partition\_value* [, ...] SET LOCATION { 's3://bucket/folder' | 's3://bucket/manifest\_file' } )**

A clause that sets a new location for one or more partition columns.

ADD [ IF NOT EXISTS ] PARTITION ( *partition\_column=partition\_value* [, ...] ) LOCATION { 's3://bucket/folder' | 's3://bucket/manifest\_file' } [, ... ]

A clause that adds one or more partitions. You can specify multiple PARTITION clauses using a single ALTER TABLE ... ADD statement.

**Note**

If you use the AWS Glue catalog, you can add up to 100 partitions using a single ALTER TABLE statement.

The IF NOT EXISTS clause indicates that if the specified partition already exists, the command should make no changes. It also indicates that the command should return a message that the partition exists, rather than terminating with an error. This clause is useful when scripting, so the script doesn't fail if ALTER TABLE tries to add a partition that already exists.

DROP PARTITION (*partition\_column=partition\_value* [, ...] )

A clause that drops the specified partition. Dropping a partition alters only the external table metadata. The data on Amazon S3 isn't affected.

ROW LEVEL SECURITY { ON | OFF }

A clause that turns on or off row-level security for a table.

When row-level security is turned on for a table, you can only read the rows that the row-level security policy permits you to access. When there isn't any policy granting you access to the table, you can't see any rows from the table. For more information, see [Row-level security \(p. 547\)](#).

FOR DATASHARES

A clause that determines whether a table is RLS-protected over datashares. By default, a RLS-protected table is also RLS-protected over datashares.

## Examples

For examples that show how to use the ALTER TABLE command, see the following.

- [ALTER TABLE examples \(p. 649\)](#)
- [ALTER EXTERNAL TABLE examples \(p. 653\)](#)
- [ALTER TABLE ADD and DROP COLUMN examples \(p. 654\)](#)

## ALTER TABLE examples

The following examples demonstrate basic usage of the ALTER TABLE command.

### Rename a table or view

The following command renames the USERS table to USERS\_BKUP:

```
alter table users
rename to users_bkup;
```

You can also use this type of command to rename a view.

### Change the owner of a table or view

The following command changes the VENUE table owner to the user DWUSER:

```
alter table venue
owner to dwuser;
```

The following commands create a view, then change its owner:

```
create view vdate as select * from date;
alter table vdate owner to vuser;
```

## Rename a column

The following command renames the VENUESEATS column in the VENUE table to VENUESIZE:

```
alter table venue
rename column venueSeats to venuesize;
```

## Drop a table constraint

To drop a table constraint, such as a primary key, foreign key, or unique constraint, first find the internal name of the constraint. Then specify the constraint name in the ALTER TABLE command. The following example finds the constraints for the CATEGORY table, then drops the primary key with the name category\_pkey.

```
select constraint_name, constraint_type
from information_schema.table_constraints
where constraint_schema = 'public'
and table_name = 'category';

constraint_name | constraint_type
-----+-----
category_pkey | PRIMARY KEY

alter table category
drop constraint category_pkey;
```

## Alter a VARCHAR column

To conserve storage, you can define a table initially with VARCHAR columns with the minimum size needed for your current data requirements. Later, to accommodate longer strings, you can alter the table to increase the size of the column.

The following example increases the size of the EVENTNAME column to VARCHAR(300).

```
alter table event alter column eventname type varchar(300);
```

## Alter the compression encoding for a column

You can alter the compression encoding of a column. Below, you can find a set of examples demonstrating this approach. The table definition for these examples is as follows.

```
create table t1(c0 int encode lzo, c1 bigint encode zstd, c2 varchar(16) encode lzo, c3
varchar(32) encode zstd);
```

The following statement alters the compression encoding for column c0 from LZO encoding to AZ64 encoding.

```
alter table t1 alter column c0 encode az64;
```

The following statement alters the compression encoding for column c1 from Zstandard encoding to AZ64 encoding.

```
alter table t1 alter column c1 encode az64;
```

The following statement alters the compression encoding for column c2 from LZO encoding to Byte-dictionary encoding.

```
alter table t1 alter column c2 encode bytedict;
```

The following statement alters the compression encoding for column c3 from Zstandard encoding to Runlength encoding.

```
alter table t1 alter column c3 encode runlength;
```

## Alter a DISTSTYLE KEY DISTKEY column

The following examples show how to change the DISTSTYLE and DISTKEY of a table.

Create a table with an EVEN distribution style. The SVV\_TABLE\_INFO view shows that the DISTSTYLE is EVEN.

```
create table inventory(
 inv_date_sk int4 not null ,
 inv_item_sk int4 not null ,
 inv_warehouse_sk int4 not null ,
 inv_quantity_on_hand int4
) diststyle even;

Insert into inventory values(1,1,1,1);

select "table", "diststyle" from svv_table_info;

table	diststyle
inventory	EVEN


```

Alter the table DISTKEY to inv\_warehouse\_sk. The SVV\_TABLE\_INFO view shows the inv\_warehouse\_sk column as the resulting distribution key.

```
alter table inventory alter diststyle key distkey inv_warehouse_sk;

select "table", "diststyle" from svv_table_info;

table	diststyle
inventory	KEY(inv_warehouse_sk)


```

Alter the table DISTKEY to inv\_item\_sk. The SVV\_TABLE\_INFO view shows the inv\_item\_sk column as the resulting distribution key.

```
alter table inventory alter distkey inv_item_sk;

select "table", "diststyle" from svv_table_info;
```

| table     | diststyle        |
|-----------|------------------|
| inventory | KEY(inv_item_sk) |

## Alter a table to DISTSTYLE ALL

The following examples show how to change a table to DISTSTYLE ALL.

Create a table with an EVEN distribution style. The SVV\_TABLE\_INFO view shows that the DISTSTYLE is EVEN.

```
create table inventory(
 inv_date_sk int4 not null ,
 inv_item_sk int4 not null ,
 inv_warehouse_sk int4 not null ,
 inv_quantity_on_hand int4
) diststyle even;

Insert into inventory values(1,1,1,1);

select "table", "diststyle" from svv_table_info;

table | diststyle
-----+-----
inventory | EVEN
```

Alter the table DISTSTYLE to ALL. The SVV\_TABLE\_INFO view shows the changed DISTSYTLE.

```
alter table inventory alter diststyle all;

select "table", "diststyle" from svv_table_info;

table | diststyle
-----+-----
inventory | ALL
```

## Alter a table SORTKEY

You can alter a table to have a compound sort key or no sort key.

In the following table definition, table t1 is defined with an interleaved sortkey.

```
create table t1 (c0 int, c1 int) interleaved sortkey(c0, c1);
```

The following command alters the table from an interleaved sort key to a compound sort key.

```
alter table t1 alter sortkey(c0, c1);
```

The following command alters the table to remove the interleaved sort key.

```
alter table t1 alter sortkey none;
```

In the following table definition, table t1 is defined with column c0 as a sortkey.

```
create table t1 (c0 int, c1 int) sortkey(c0);
```

The following command alters the table t1 to a compound sort key.

```
alter table t1 alter sortkey(c0, c1);
```

## Alter a table to ENCODE AUTO

The following example shows how to alter a table to ENCODE AUTO.

The table definition for this example follows. Column c0 is defined with the encoding type AZ64, and column c1 is defined with the encoding type LZO.

```
create table t1(c0 int encode AZ64, c1 varchar encode LZO);
```

For this table, the following statement alters the encoding to AUTO.

```
alter table t1 alter encode auto;
```

The following example shows how to alter a table to remove the ENCODE AUTO setting.

The table definition for this example follows. The table columns are defined without encoding. In this case, the encoding defaults to ENCODE AUTO.

```
create table t2(c0 int, c1 varchar);
```

For this table, the following statement alters the encoding of column c0 to LZO. The table encoding is no longer set to ENCODE AUTO.

```
alter table t2 alter column c0 encode lzo;;
```

## Alter row-level security control

The following command turns RLS off for the table:

```
ALTER TABLE ticket_category_redshift ROW LEVEL SECURITY OFF;
```

The following command turns RLS on for the table:

```
ALTER TABLE ticket_category_redshift ROW LEVEL SECURITY ON FOR DATASHARES;
```

## ALTER EXTERNAL TABLE examples

The following example sets the numRows table property for the SPECTRUM.SALES external table to 170,000 rows.

```
alter table spectrum.sales
set table properties ('numRows'='170000');
```

The following example changes the location for the SPECTRUM.SALES external table.

```
alter table spectrum.sales
set location 's3://awssampledbuswest2/ticket/spectrum/sales/';
```

The following example changes the format for the SPECTRUM.SALES external table to Parquet.

```
alter table spectrum.sales
set file format parquet;
```

The following example adds one partition for the table SPECTRUM.SALES\_PART.

```
alter table spectrum.sales_part
add if not exists partition(saledate='2008-01-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-01/';
```

The following example adds three partitions for the table SPECTRUM.SALES\_PART.

```
alter table spectrum.sales_part add if not exists
partition(saledate='2008-01-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-01/'
partition(saledate='2008-02-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-02/'
partition(saledate='2008-03-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-03/';
```

The following example alters SPECTRUM.SALES\_PART to drop the partition with saledate='2008-01-01'.

```
alter table spectrum.sales_part
drop partition(saledate='2008-01-01');
```

The following example sets a new Amazon S3 path for the partition with saledate='2008-01-01'.

```
alter table spectrum.sales_part
partition(saledate='2008-01-01')
set location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/
saledate=2008-01-01/';
```

The following example changes the name of sales\_date to transaction\_date.

```
alter table spectrum.sales rename column sales_date to transaction_date;
```

The following example sets the column mapping to position mapping for an external table that uses optimized row columnar (ORC) format.

```
alter table spectrum.orc_example
set table properties('orc.schema.resolution'='position');
```

The following example sets the column mapping to name mapping for an external table that uses ORC format.

```
alter table spectrum.orc_example
set table properties('orc.schema.resolution'='name');
```

## ALTER TABLE ADD and DROP COLUMN examples

The following examples demonstrate how to use ALTER TABLE to add and then drop a basic table column and also how to drop a column with a dependent object.

## ADD then DROP a basic column

The following example adds a standalone FEEDBACK\_SCORE column to the USERS table. This column simply contains an integer, and the default value for this column is NULL (no feedback score).

First, query the PG\_TABLE\_DEF catalog table to view the schema of the USERS table:

| column        | type                   | encoding | distkey | sortkey |
|---------------|------------------------|----------|---------|---------|
| userid        | integer                | delta    | true    | 1       |
| username      | character(8)           | lzo      | false   | 0       |
| firstname     | character varying(30)  | text32k  | false   | 0       |
| lastname      | character varying(30)  | text32k  | false   | 0       |
| city          | character varying(30)  | text32k  | false   | 0       |
| state         | character(2)           | bytedict | false   | 0       |
| email         | character varying(100) | lzo      | false   | 0       |
| phone         | character(14)          | lzo      | false   | 0       |
| likesports    | boolean                | none     | false   | 0       |
| liketheatre   | boolean                | none     | false   | 0       |
| likeconcerts  | boolean                | none     | false   | 0       |
| likejazz      | boolean                | none     | false   | 0       |
| likeclassical | boolean                | none     | false   | 0       |
| likeopera     | boolean                | none     | false   | 0       |
| likerock      | boolean                | none     | false   | 0       |
| likevegas     | boolean                | none     | false   | 0       |
| likebroadway  | boolean                | none     | false   | 0       |
| likemusicals  | boolean                | none     | false   | 0       |

Now add the feedback\_score column:

```
alter table users
add column feedback_score int
default NULL;
```

Select the FEEDBACK\_SCORE column from USERS to verify that it was added:

```
select feedback_score from users limit 5;

feedback_score

NULL
NULL
NULL
NULL
NULL
```

Drop the column to reinstate the original DDL:

```
alter table users drop column feedback_score;
```

## Dropping a column with a dependent object

The following example drops a column that has a dependent object. As a result, the dependent object is also dropped.

To start, add the FEEDBACK\_SCORE column to the USERS table again:

```
alter table users
add column feedback_score int
```

```
default NULL;
```

Next, create a view from the USERS table called USERS\_VIEW:

```
create view users_view as select * from users;
```

Now, try to drop the FEEDBACK\_SCORE column from the USERS table. This DROP statement uses the default behavior (RESTRICT):

```
alter table users drop column feedback_score;
```

Amazon Redshift displays an error message that the column can't be dropped because another object depends on it.

Try dropping the FEEDBACK\_SCORE column again, this time specifying CASCADE to drop all dependent objects:

```
alter table users
drop column feedback_score cascade;
```

## ALTER TABLE APPEND

Appends rows to a target table by moving data from an existing source table. Data in the source table is moved to matching columns in the target table. Column order doesn't matter. After data is successfully appended to the target table, the source table is empty. ALTER TABLE APPEND is usually much faster than a similar [CREATE TABLE AS \(p. 838\)](#) or [INSERT \(p. 902\)](#) INTO operation because data is moved, not duplicated.

### Note

ALTER TABLE APPEND moves data blocks between the source table and the target table. To improve performance, ALTER TABLE APPEND doesn't compact storage as part of the append operation. As a result, storage usage increases temporarily. To reclaim the space, run a [VACUUM \(p. 1015\)](#) operation.

Columns with the same names must also have identical column attributes. If either the source table or the target table contains columns that don't exist in the other table, use the IGNOREEXTRA or FILLTARGET parameters to specify how extra columns should be managed.

You can't append an identity column. If both tables include an identity column, the command fails. If only one table has an identity column, include the FILLTARGET or IGNOREEXTRA parameter. For more information, see [ALTER TABLE APPEND usage notes \(p. 657\)](#).

You can append a GENERATED BY DEFAULT AS IDENTITY column. You can update columns defined as GENERATED BY DEFAULT AS IDENTITY with values that you supply. For more information, see [ALTER TABLE APPEND usage notes \(p. 657\)](#).

Both the source table and the target table must be permanent tables. Both tables must use the same distribution style and distribution key, if one was defined. If the tables are sorted, both tables must use the same sort style and define the same columns as sort keys.

An ALTER TABLE APPEND command automatically commits immediately upon completion of the operation. It can't be rolled back. You can't run ALTER TABLE APPEND within a transaction block (BEGIN ... END). For more information about transactions, see [Serializable isolation \(p. 119\)](#).

## Required privileges

Depending on the ALTER TABLE APPEND command, one of the following privileges is required:

- Superuser
- Users with the ALTER TABLE system privilege
- Users with DELETE and SELECT privileges on the source table, and INSERT privilege on the target table

## Syntax

```
ALTER TABLE target_table_name APPEND FROM source_table_name
[IGNOREEXTRA | FILLTARGET]
```

## Parameters

*target\_table\_name*

The name of the table to which rows are appended. Either specify just the name of the table or use the format *schema\_name.table\_name* to use a specific schema. The target table must be an existing permanent table.

FROM *source\_table\_name*

The name of the table that provides the rows to be appended. Either specify just the name of the table or use the format *schema\_name.table\_name* to use a specific schema. The source table must be an existing permanent table.

IGNOREEXTRA

A keyword that specifies that if the source table includes columns that are not present in the target table, data in the extra columns should be discarded. You can't use IGNOREEXTRA with FILLTARGET.

FILLTARGET

A keyword that specifies that if the target table includes columns that are not present in the source table, the columns should be filled with the [DEFAULT \(p. 826\)](#) column value, if one was defined, or NULL. You can't use IGNOREEXTRA with FILLTARGET.

## ALTER TABLE APPEND usage notes

ALTER TABLE APPEND moves only identical columns from the source table to the target table. Column order doesn't matter.

If either the source table or the target table contains extra columns, use either FILLTARGET or IGNOREEXTRA according to the following rules:

- If the source table contains columns that don't exist in the target table, include IGNOREEXTRA. The command ignores the extra columns in the source table.
- If the target table contains columns that don't exist in the source table, include FILLTARGET. The command fills the extra columns in the target table with either the default column value or IDENTITY value, if one was defined, or NULL.
- If both the source table and the target table contain extra columns, the command fails. You can't use both FILLTARGET and IGNOREEXTRA.

If a column with the same name but different attributes exists in both tables, the command fails. Like-named columns must have the following attributes in common:

- Data type
- Column size

- Compression encoding
- Not null
- Sort style
- Sort key columns
- Distribution style
- Distribution key columns

You can't append an identity column. If both the source table and the target table have identity columns, the command fails. If only the source table has an identity column, include the IGNOREEXTRA parameter so that the identity column is ignored. If only the target table has an identity column, include the FILLTARGET parameter so that the identity column is populated according to the IDENTITY clause defined for the table. For more information, see [DEFAULT \(p. 826\)](#).

You can append a default identity column with the ALTER TABLE APPEND statement. For more information, see [CREATE TABLE \(p. 823\)](#).

## ALTER TABLE APPEND examples

Suppose your organization maintains a table, SALES\_MONTHLY, to capture current sales transactions. You want to move data from the transaction table to the SALES table, every month.

You can use the following INSERT INTO and TRUNCATE commands to accomplish the task.

```
insert into sales (select * from sales_monthly);
truncate sales_monthly;
```

However, you can perform the same operation much more efficiently by using an ALTER TABLE APPEND command.

First, query the [PG\\_TABLE\\_DEF \(p. 1677\)](#) system catalog table to verify that both tables have the same columns with identical column attributes.

```
select trim(tablename) as table, "column", trim(type) as type,
encoding, distkey, sortkey, "notnull"
from pg_table_def where tablename like 'sales%';

table	column	type	encoding	distkey	sortkey	
sales	salesid	integer	lzo	false	0	true
sales	listid	integer	none	true	1	true
sales	sellerid	integer	none	false	2	true
sales	buyerid	integer	lzo	false	0	true
sales	eventid	integer	mostly16	false	0	true
sales	dateid	smallint	lzo	false	0	true
sales	qtysold	smallint	mostly8	false	0	true
sales	pricepaid	numeric(8,2)	delta32k	false	0	
sales	commission	numeric(8,2)	delta32k	false	0	
sales	saletime	timestamp without time zone	lzo	false	0	
salesmonth	salesid	integer	lzo	false	0	true
salesmonth	listid	integer	none	true	1	true
salesmonth	sellerid	integer	none	false	2	true
salesmonth	buyerid	integer	lzo	false	0	true
salesmonth	eventid	integer	mostly16	false	0	true


```

|                                                     |                            |
|-----------------------------------------------------|----------------------------|
| salesmonth   dateid   smallint                      | lzo   false   0   true     |
| salesmonth   qtysold   smallint                     | mostly8   false   0   true |
| salesmonth   pricepaid   numeric(8,2)               | delta32k   false   0       |
| false                                               |                            |
| salesmonth   commission   numeric(8,2)              | delta32k   false   0       |
| false                                               |                            |
| salesmonth   saletime   timestamp without time zone | lzo   false   0            |
| false                                               |                            |

Next, look at the size of each table.

```
select count(*) from sales_monthly;
count

2000
(1 row)

select count(*) from sales;
count

412,214
(1 row)
```

Now run the following ALTER TABLE APPEND command.

```
alter table sales append from sales_monthly;
```

Look at the size of each table again. The SALES\_MONTHLY table now has 0 rows, and the SALES table has grown by 2000 rows.

```
select count(*) from sales_monthly;
count

0
(1 row)

select count(*) from sales;
count

414214
(1 row)
```

If the source table has more columns than the target table, specify the IGNOREEXTRA parameter. The following example uses the IGNOREEXTRA parameter to ignore extra columns in the SALES\_LISTING table when appending to the SALES table.

```
alter table sales append from sales_listing ignoreextra;
```

If the target table has more columns than the source table, specify the FILLTARGET parameter. The following example uses the FILLTARGET parameter to populate columns in the SALES\_REPORT table that don't exist in the SALES\_MONTH table.

```
alter table sales_report append from sales_month filltarget;
```

## ALTER USER

Changes a database user.

## Required privileges

Following are required privileges for ALTER USER:

- Superuser
- Users with the ALTER USER privilege
- Current user who wants to change their own password

## Syntax

```
ALTER USER username [WITH] option [, ...]
where option is
CREATEDB | NOCREATEDB
| CREATEUSER | NOCREATEUSER
| SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }
| PASSWORD { 'password' | 'md5hash' | DISABLE }
[VALID UNTIL 'expiration_date']
| RENAME TO new_name |
| CONNECTION LIMIT { limit | UNLIMITED }
| SESSION TIMEOUT limit | RESET SESSION TIMEOUT
| SET parameter { TO | = } { value | DEFAULT }
| RESET parameter
| EXTERNALID external_id
```

## Parameters

*username*

Name of the user.

WITH

Optional keyword.

CREATEDB | NOCREATEDB

The CREATEDB option allows the user to create new databases. NOCREATEDB is the default.

CREATEUSER | NOCREATEUSER

The CREATEUSER option creates a superuser with all database privileges, including CREATE USER.

The default is NOCREATEUSER. For more information, see [superuser \(p. 537\)](#).

SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }

A clause that specifies the level of access that the user has to the Amazon Redshift system tables and views.

If RESTRICTED is specified, the user can see only the rows generated by that user in user-visible system tables and views. The default is RESTRICTED.

If UNRESTRICTED is specified, the user can see all rows in user-visible system tables and views, including rows generated by another user. UNRESTRICTED doesn't give a regular user access to superuser-visible tables. Only superusers can see superuser-visible tables.

### Note

Giving a user unrestricted access to system tables gives the user visibility to data generated by other users. For example, STL\_QUERY and STL\_QUERYTEXT contain the full text of

INSERT, UPDATE, and DELETE statements, which might contain sensitive user-generated data.

All rows in SVV\_TRANSACTIONS are visible to all users.

For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

PASSWORD { 'password' | 'md5hash' | DISABLE }

Sets the user's password.

By default, users can change their own passwords, unless the password is disabled. To disable a user's password, specify DISABLE. When a user's password is disabled, the password is deleted from the system and the user can log on only using temporary AWS Identity and Access Management (IAM) user credentials. For more information, see [Using IAM authentication to generate database user credentials](#). Only a superuser can enable or disable passwords. You can't disable a superuser's password. To enable a password, run ALTER USER and specify a password.

For details about using the PASSWORD parameter, see [CREATE USER \(p. 846\)](#).

VALID UNTIL '*expiration\_date*'

Specifies that the password has an expiration date. Use the value 'infinity' to avoid having an expiration date. The valid data type for this parameter is timestamp.

RENAME TO

Renames the user.

*new\_name*

New name of the user. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

**Important**

When you rename a user, you must also change the user's password. The user name is used as part of the password encryption, so when a user is renamed, the password is cleared. The user will not be able to log on until the password is reset. For example:

```
alter user newuser password 'EXAMPLENewPassword11';
```

CONNECTION LIMIT { *limit* | UNLIMITED }

The maximum number of database connections the user is permitted to have open concurrently. The limit isn't enforced for superusers. Use the UNLIMITED keyword to permit the maximum number of concurrent connections. A limit on the number of connections for each database might also apply. For more information, see [CREATE DATABASE \(p. 745\)](#). The default is UNLIMITED. To view current connections, query the [STV\\_SESSIONS \(p. 1597\)](#) system view.

**Note**

If both user and database connection limits apply, an unused connection slot must be available that is within both limits when a user attempts to connect.

SESSION TIMEOUT *limit* | RESET SESSION TIMEOUT

The maximum time in seconds that a session remains inactive or idle. The range is 60 seconds (one minute) to 1,728,000 seconds (20 days). If no session timeout is set for the user, the cluster setting applies. For more information, see [Quotas and limits in Amazon Redshift](#) in the *Amazon Redshift Management Guide*.

When you set the session timeout, it's applied to new sessions only.

To view information about active user sessions, including the start time, user name, and session timeout, query the [STV\\_SESSIONS \(p. 1597\)](#) system view. To view information about user-session

history, query the [STL\\_SESSIONS \(p. 1557\)](#) view. To retrieve information about database users, including session-timeout values, query the [SVL\\_USER\\_INFO \(p. 1669\)](#) view.

**SET**

Sets a configuration parameter to a new default value for all sessions run by the specified user.

**RESET**

Resets a configuration parameter to the original default value for the specified user.

*parameter*

Name of the parameter to set or reset.

*value*

New value of the parameter.

**DEFAULT**

Sets the configuration parameter to the default value for all sessions run by the specified user.

**EXTERNALID** *external\_id*

The identifier for the user, which is associated with an identity provider. The user must have their password disabled. For more information, see [Native identity provider \(IdP\) federation for Amazon Redshift](#).

## Usage notes

When using AWS Identity and Access Management (IAM) authentication to create database user credentials, you might want to create a superuser that is able to log on only using temporary credentials. You can't disable a superuser's password, but you can create an unknown password using a randomly generated MD5 hash string.

```
alter user iam_superuser password 'md51234567890123456780123456789012';
```

When you set the [search\\_path \(p. 1696\)](#) parameter with the ALTER USER command, the modification takes effect on the specified user's next login. If you want to change the search\_path value for the current user and session, use a SET command.

## Examples

The following example gives the user ADMIN the privilege to create databases:

```
alter user admin createdb;
```

The following example sets the password of the user ADMIN to adminPass9 and sets an expiration date and time for the password:

```
alter user admin password 'adminPass9'
valid until '2017-12-31 23:59';
```

The following example renames the user ADMIN to SYSADMIN:

```
alter user admin rename to sysadmin;
```

The following example updates the idle-session timeout for a user to 300 seconds.

```
ALTER USER dbuser SESSION TIMEOUT 300;
```

Resets the user's idle-session timeout. When you reset it, the cluster setting applies. You must be a database superuser to run this command. For more information, see [Quotas and limits in Amazon Redshift](#) in the *Amazon Redshift Management Guide*.

```
ALTER USER dbuser RESET SESSION TIMEOUT;
```

The following example updates the external ID for a user named bob. The namespace is myco\_aad. If the namespace isn't associated with a registered identity provider, it results in an error.

```
ALTER USER myco_aad:bob EXTERNALID "ABC123" PASSWORD DISABLE;
```

## ANALYZE

Updates table statistics for use by the query planner.

### Required privileges

Following are required privileges for ANALYZE:

- Superuser
- Users with the ANALYZE privilege
- Owner of the relation
- Database owner whom the table is shared to

## Syntax

```
ANALYZE [VERBOSE]
[[table_name [(column_name [, ...])]]
[PREDICATE COLUMNS | ALL COLUMNS]
```

## Parameters

### VERBOSE

A clause that returns progress information messages about the ANALYZE operation. This option is useful when you don't specify a table.

### table\_name

You can analyze specific tables, including temporary tables. You can qualify the table with its schema name. You can optionally specify a table\_name to analyze a single table. You can't specify more than one table\_name with a single ANALYZE table\_name statement. If you don't specify a table\_name value, all of the tables in the currently connected database are analyzed, including the persistent tables in the system catalog. Amazon Redshift skips analyzing a table if the percentage of rows that have changed since the last ANALYZE is lower than the analyze threshold. For more information, see [Analyze threshold \(p. 664\)](#).

You don't need to analyze Amazon Redshift system tables (STL and STV tables).

### column\_name

If you specify a table\_name, you can also specify one or more columns in the table (as a column-separated list within parentheses). If a column list is specified, only the listed columns are analyzed.

## PREDICATE COLUMNS | ALL COLUMNS

Clauses that indicate whether ANALYZE should include only predicate columns. Specify PREDICATE COLUMNS to analyze only columns that have been used as predicates in previous queries or are likely candidates to be used as predicates. Specify ALL COLUMNS to analyze all columns. The default is ALL COLUMNS.

A column is included in the set of predicate columns if any of the following is true:

- The column has been used in a query as a part of a filter, join condition, or group by clause.
- The column is a distribution key.
- The column is part of a sort key.

If no columns are marked as predicate columns, for example because the table has not yet been queried, all of the columns are analyzed even when PREDICATE COLUMNS is specified. For more information about predicate columns, see [Analyzing tables \(p. 105\)](#).

## Usage notes

Amazon Redshift automatically runs ANALYZE on tables that you create with the following commands:

- CREATE TABLE AS
- CREATE TEMP TABLE AS
- SELECT INTO

You can't analyze an external table.

You don't need to run the ANALYZE command on these tables when they are first created. If you modify them, you should analyze them in the same way as other tables.

## Analyze threshold

To reduce processing time and improve overall system performance, Amazon Redshift skips ANALYZE for a table if the percentage of rows that have changed since the last ANALYZE command run is lower than the analyze\_threshold\_percent specified by the [analyze\\_threshold\\_percent \(p. 1685\)](#) parameter. By default, analyze\_threshold\_percent is 10. To change analyze\_threshold\_percent for the current session, run the [SET \(p. 977\)](#) command. The following example changes analyze\_threshold\_percent to 20 percent.

```
set analyze_threshold_percent to 20;
```

To analyze tables when only a small number of rows have changed, set analyze\_threshold\_percent to an arbitrarily small number. For example, if you set analyze\_threshold\_percent to 0.01, then a table with 100,000,000 rows aren't skipped if at least 10,000 rows have changed.

```
set analyze_threshold_percent to 0.01;
```

If ANALYZE skips a table because it doesn't meet the analyze threshold, Amazon Redshift returns the following message.

```
ANALYZE SKIP
```

To analyze all tables even if no rows have changed, set analyze\_threshold\_percent to 0.

To view the results of ANALYZE operations, query the [STL\\_ANALYZE \(p. 1499\)](#) system table.

For more information about analyzing tables, see [Analyzing tables \(p. 105\)](#).

## Examples

Analyze all of the tables in the TICKIT database and return progress information.

```
analyze verbose;
```

Analyze the LISTING table only.

```
analyze listing;
```

Analyze the VENUEID and VENUENAME columns in the VENUE table.

```
analyze venue(venueid, venuename);
```

Analyze only predicate columns in the VENUE table.

```
analyze venue predicate columns;
```

## ANALYZE COMPRESSION

Performs compression analysis and produces a report with the suggested compression encoding for the tables analyzed. For each column, the report includes an estimate of the potential reduction in disk space compared to the current encoding.

## Syntax

```
ANALYZE COMPRESSION
[[table_name]
[(column_name [, ...])]
[COMPROWS numrows]
```

## Parameters

*table\_name*

You can analyze compression for specific tables, including temporary tables. You can qualify the table with its schema name. You can optionally specify a *table\_name* to analyze a single table. If you don't specify a *table\_name*, all of the tables in the currently connected database are analyzed. You can't specify more than one *table\_name* with a single ANALYZE COMPRESSION statement.

*column\_name*

If you specify a *table\_name*, you can also specify one or more columns in the table (as a column-separated list within parentheses).

**COMPROWS**

Number of rows to be used as the sample size for compression analysis. The analysis is run on rows from each data slice. For example, if you specify COMPROWS 1000000 (1,000,000) and the system contains 4 total slices, no more than 250,000 rows per slice are read and analyzed. If COMPROWS isn't specified, the sample size defaults to 100,000 per slice. Values of COMPROWS lower than the default of 100,000 rows per slice are automatically upgraded to the default value. However,

compression analysis doesn't produce recommendations if the amount of data in the table is insufficient to produce a meaningful sample. If the COMPROWS number is greater than the number of rows in the table, the ANALYZE COMPRESSION command still proceeds and runs the compression analysis against all of the available rows.

#### *numrows*

Number of rows to be used as the sample size for compression analysis. The accepted range for *numrows* is a number between 1000 and 1000000000 (1,000,000,000).

## Usage notes

ANALYZE COMPRESSION acquires an exclusive table lock, which prevents concurrent reads and writes against the table. Only run the ANALYZE COMPRESSION command when the table is idle.

Run ANALYZE COMPRESSION to get recommendations for column encoding schemes, based on a sample of the table's contents. ANALYZE COMPRESSION is an advisory tool and doesn't modify the column encodings of the table. You can apply the suggested encoding by recreating the table or by creating a new table with the same schema. Recreating an uncompressed table with appropriate encoding schemes can significantly reduce its on-disk footprint. This approach saves disk space and improves query performance for I/O-bound workloads.

ANALYZE COMPRESSION skips the actual analysis phase and directly returns the original encoding type on any column that is designated as a SORTKEY. It does this because range-restricted scans might perform poorly when SORTKEY columns are compressed much more highly than other columns.

## Examples

The following example shows the encoding and estimated percent reduction for the columns in the LISTING table only:

```
analyze compression listing;

Table | Column | Encoding | Est_reduction_pct
-----+-----+-----+-----
listing | listid | delta | 75.00
listing | sellerid | delta32k | 38.14
listing | eventid | delta32k | 5.88
listing | dateid | zstd | 31.73
listing | numtickets | zstd | 38.41
listing | priceperticket | zstd | 59.48
listing | totalprice | zstd | 37.90
listing | listtime | zstd | 13.39
```

The following example analyzes the QTYSOLD, COMMISSION, and SALETIME columns in the SALES table.

```
analyze compression sales(qtysold, commission, saletime);

Table | Column | Encoding | Est_reduction_pct
-----+-----+-----+-----
sales | salesid | N/A | 0.00
sales | listid | N/A | 0.00
sales | sellerid | N/A | 0.00
sales | buyerid | N/A | 0.00
sales | eventid | N/A | 0.00
sales | dateid | N/A | 0.00
sales | qtysold | zstd | 67.14
sales | pricepaid | N/A | 0.00
```

|                           |       |
|---------------------------|-------|
| sales   commission   zstd | 13.94 |
| sales   saletime   zstd   | 13.38 |

## ATTACH MASKING POLICY

Attaches an existing dynamic data masking policy to a column. For more information on dynamic data masking, see [Dynamic data masking \(p. 555\)](#).

Superusers and users or roles that have the sys:secadmin role can attach a masking policy.

### Syntax

```
ATTACH MASKING POLICY policy_name
 ON { table_name }
 (output_columns_names) [USING (input_column_names)]
 TO { user_name | ROLE role_name | PUBLIC }
 [PRIORITY priority];
```

### Parameters

*policy\_name*

The name of the masking policy to attach.

*table\_name*

The name of the table to attach the masking policy to.

*output\_column\_names*

The names of the columns that the masking policy will apply to.

*input\_column\_names*

The names of the columns that the masking policy will take as input. This parameter is optional. If not specified, the masking policy uses the *output\_column\_names* as inputs.

*user\_name*

The name of the user to whom the masking policy will attach. You can't attach two policies to the same column/user pair. You can attach a policy to a user and another policy to the user's role. In this case, the policy with the higher priority applies.

You can only set one of *user\_name*, *role\_name*, and PUBLIC in a single ATTACH MASKING POLICY command.

*role\_name*

The name of the role to which the masking policy will attach. You can't attach two policies to the same column/role pair. You can attach a policy to a user and another policy to the user's role. In this case, the policy with the higher priority applies.

You can only set one of *user\_name*, *role\_name*, and PUBLIC in a single ATTACH MASKING POLICY command.

*PUBLIC*

Attaches the masking policy to all users accessing the table. You must give other masking policies attached to specific column/user or column/role pairs a higher priority than the PUBLIC policy for them to apply.

You can only set one of user\_name, role\_name, and PUBLIC in a single ATTACH MASKING POLICY command.

*priority*

The priority of the masking policy. When multiple masking policies apply to a given user's query, the highest priority policy applies.

You can't apply a policy to a column with the same priority as another policy attached to that column, even if they're for different roles. This field is optional. If you don't specify a priority, the masking policy defaults to attaching with a priority of 0.

## ATTACH RLS POLICY

Attach a row-level security policy on a table to one or more users or roles.

Superusers and users or roles that have the sys:secadmin role can attach a policy.

### Syntax

```
ATTACH RLS POLICY policy_name ON [TABLE] table_name [, ...]
TO { user_name | ROLE role_name | PUBLIC } [, ...]
```

### Parameters

*policy\_name*

The name of the policy.

ON [TABLE] *table\_name* [, ...]

The table or view that the row-level security policy is attached to.

TO { *user\_name* | ROLE *role\_name* | PUBLIC } [, ...]

Specifies whether the policy is attached to one or more specified users or roles.

### Usage notes

When working with the ATTACH RLS POLICY statement, observe the following:

- The table being attached should have all the columns listed in the WITH clause of the policy creation statement.
- Amazon Redshift RLS doesn't support attaching RLS policies to the following objects: catalog tables, cross-database relations, external tables, regular views, late-binding views, temporary tables, and lookup tables for RLS policies.
- You can't attach a RLS policy to superusers or to a sys:secadmin.

### Examples

The following example attaches a policy on a table to a role.

```
ATTACH RLS POLICY policy_concerts ON ticket_category_redshift TO ROLE analyst, ROLE
dbadmin;
```

## BEGIN

Starts a transaction. Synonymous with START TRANSACTION.

A transaction is a single, logical unit of work, whether it consists of one command or multiple commands. In general, all commands in a transaction run on a snapshot of the database whose starting time is determined by the value set for the `transaction_snapshot_begin` system configuration parameter.

By default, individual Amazon Redshift operations (queries, DDL statements, loads) are automatically committed to the database. If you want to suspend the commit for an operation until subsequent work is completed, you need to open a transaction with the BEGIN statement, then run the required commands, then close the transaction with a [COMMIT \(p. 676\)](#) or [END \(p. 879\)](#) statement. If necessary, you can use a [ROLLBACK \(p. 927\)](#) statement to stop a transaction that is in progress. An exception to this behavior is the [TRUNCATE \(p. 990\)](#) command, which commits the transaction in which it is run and can't be rolled back.

## Syntax

```
BEGIN [WORK | TRANSACTION] [ISOLATION LEVEL option] [READ WRITE | READ ONLY]
```

```
START TRANSACTION [ISOLATION LEVEL option] [READ WRITE | READ ONLY]
```

Where *option* is

```
SERIALIZABLE
| READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
```

Note: READ UNCOMMITTED, READ COMMITTED, and REPEATABLE READ have no operational impact and map to SERIALIZABLE in Amazon Redshift. You can see database isolation levels on your cluster by querying the `stv_db_isolation_level` table.

## Parameters

WORK

Optional keyword.

TRANSACTION

Optional keyword; WORK and TRANSACTION are synonyms.

ISOLATION LEVEL SERIALIZABLE

Serializable isolation is supported by default, so the behavior of the transaction is the same whether or not this syntax is included in the statement. For more information, see [Managing concurrent write operations \(p. 118\)](#). No other isolation levels are supported.

### Note

The SQL standard defines four levels of transaction isolation to prevent *dirty reads* (where a transaction reads data written by a concurrent uncommitted transaction), *nonrepeatable reads* (where a transaction re-reads data it read previously and finds that data was changed by another transaction that committed since the initial read), and *phantom reads* (where a transaction re-runs a query, returns a set of rows that satisfy a search condition, and then finds that the set of rows has changed because of another recently committed transaction):

- Read uncommitted: Dirty reads, nonrepeatable reads, and phantom reads are possible.
- Read committed: Nonrepeatable reads and phantom reads are possible.

- Repeatable read: Phantom reads are possible.
  - Serializable: Prevents dirty reads, nonrepeatable reads, and phantom reads.
- Though you can use any of the four transaction isolation levels, Amazon Redshift processes all isolation levels as serializable.

#### READ WRITE

Gives the transaction read and write permissions.

#### READ ONLY

Gives the transaction read-only permissions.

## Examples

The following example starts a serializable transaction block:

```
begin;
```

The following example starts the transaction block with a serializable isolation level and read and write permissions:

```
begin read write;
```

## CALL

Runs a stored procedure. The CALL command must include the procedure name and the input argument values. You must call a stored procedure by using the CALL statement. CALL can't be part of any regular queries.

## Syntax

```
CALL sp_name ([argument] [, ...])
```

## Parameters

*sp\_name*

The name of the procedure to run.

*argument*

The value of the input argument. This parameter can also be a function name, for example pg\_last\_query\_id(). You can't use queries as CALL arguments.

## Usage notes

Amazon Redshift stored procedures support nested and recursive calls, as described following. In addition, make sure your driver support is up-to-date, also described following.

### Topics

- [Nested calls \(p. 671\)](#)
- [Driver support \(p. 671\)](#)

## Nested calls

Amazon Redshift stored procedures support nested and recursive calls. The maximum number of nesting levels allowed is 16. Nested calls can encapsulate business logic into smaller procedures, which can be shared by multiple callers.

If you call a nested procedure that has output parameters, the inner procedure must define INOUT arguments. In this case, the inner procedure is passed in a nonconstant variable. OUT arguments aren't allowed. This behavior occurs because a variable is needed to hold the output of the inner call.

The relationship between inner and outer procedures is logged in the `from_sp_call` column of [SVL\\_STORED\\_PROC\\_CALL \(p. 1663\)](#).

The following example shows passing variables to a nested procedure call through INOUT arguments.

```
CREATE OR REPLACE PROCEDURE inner_proc(INOUT a int, b int, INOUT c int) LANGUAGE plpgsql
AS $$
BEGIN
 a := b * a;
 c := b * c;
END;
$$;

CREATE OR REPLACE PROCEDURE outer_proc(multiplier int) LANGUAGE plpgsql
AS $$
DECLARE
 x int := 3;
 y int := 4;
BEGIN
 DROP TABLE IF EXISTS test_tbl;
 CREATE TEMP TABLE test_tbl(a int, b varchar(256));
 CALL inner_proc(x, multiplier, y);
 insert into test_tbl values (x, y::varchar);
END;
$$;

CALL outer_proc(5);

SELECT * from test_tbl;
 a | b
----+---
 15 | 20
(1 row)
```

## Driver support

We recommend that you upgrade your Java Database Connectivity (JDBC) and Open Database Connectivity (ODBC) drivers to the latest version that has support for Amazon Redshift stored procedures.

You might be able to use your existing driver if your client tool uses driver API operations that pass through the `CALL` statement to the server. Output parameters, if any, are returned as a result set of one row.

The latest versions of Amazon Redshift JDBC and ODBC drivers have metadata support for stored procedure discovery. They also have `CallableStatement` support for custom Java applications. For more information on drivers, see [Connecting to an Amazon Redshift Cluster Using SQL Client Tools](#) in the [Amazon Redshift Management Guide](#).

The following examples show how to use different API operations of the JDBC driver for stored procedure calls.

```

void statement_example(Connection conn) throws SQLException {
 statement.execute("CALL sp_statement_example(1)");
}

void prepared_statement_example(Connection conn) throws SQLException {
 String sql = "CALL sp_prepared_statement_example(42, 84)";
 PreparedStatement pstmt = conn.prepareStatement(sql);
 pstmt.execute();
}

void callable_statement_example(Connection conn) throws SQLException {
 CallableStatement cstmt = conn.prepareCall("CALL sp_create_out_in(?,?)");
 cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
 cstmt.setInt(2, 42);
 cstmt.executeQuery();
 Integer out_value = cstmt.getInt(1);
}

```

## Examples

The following example calls the procedure name `test_sp1`.

```

call test_sp1(3,'book');
INFO: Table "tmp_tbl" does not exist and will be skipped
INFO: min_val = 3, f2 = book

```

The following example calls the procedure name `test_sp12`.

```

call test_sp2(2,'2019');

 f2 | column2
-----+-----
2019+2019+2019+2019 | 2
(1 row)

```

## CANCEL

Cancels a database query that is currently running.

The CANCEL command requires the process ID of the running query and displays a confirmation message to verify that the query was cancelled.

## Required privileges

Following are required privileges for CANCEL:

- Superuser canceling their own query
- Superuser canceling a user's query
- Users with the CANCEL privilege canceling a user's query
- User canceling their own query

## Syntax

```
CANCEL process_id ['message']
```

## Parameters

### *process\_id*

Process ID corresponding to the query that you want to cancel.

### 'message'

An optional confirmation message that displays when the query cancellation completes. If you don't specify a message, Amazon Redshift displays the default message as verification. You must enclose the message in single quotation marks.

## Usage notes

You can't cancel a query by specifying a *query ID*; you must specify the query's *process ID* (PID). You can only cancel queries currently being run by your user. Superusers can cancel all queries.

If queries in multiple sessions hold locks on the same table, you can use the [PG\\_TERMINATE\\_BACKEND \(p. 1384\)](#) function to terminate one of the sessions. Doing this forces any currently running transactions in the terminated session to release all locks and roll back the transaction. To view currently held locks, query the [STV\\_LOCKS \(p. 1586\)](#) system table.

Following certain internal events, Amazon Redshift might restart an active session and assign a new PID. If the PID has changed, you might receive the following error message.

Session <PID> does not exist. The session PID might have changed. Check the stl\_restarted\_sessions system table for details.

To find the new PID, query the [STL\\_RESTARTED\\_SESSIONS \(p. 1547\)](#) system table and filter on the oldpid column.

```
select oldpid, newpid from stl_restarted_sessions where oldpid = 1234;
```

## Examples

To cancel a currently running query, first retrieve the process ID for the query that you want to cancel. To determine the process IDs for all currently running queries, type the following command:

```
select pid, starttime, duration,
trim(user_name) as user,
trim (query) as querytxt
from stv_recents
where status = 'Running';

pid | starttime | duration | user | querytxt
---+-----+-----+-----+
802 | 2008-10-14 09:19:03.550885 | 132 | dwuser | select
venuename from venue where venuestate='FL', where venuecity not in
('Miami' , 'Orlando');
834 | 2008-10-14 08:33:49.473585 | 1250414 | dwuser | select *
from listing;
964 | 2008-10-14 08:30:43.290527 | 326179 | dwuser | select
sellerid from sales where qtysold in (8, 10);
```

Check the query text to determine which process id (PID) corresponds to the query that you want to cancel.

Type the following command to use PID 802 to cancel that query:

```
cancel 802;
```

The session where the query was running displays the following message:

```
ERROR: Query (168) cancelled on user's request
```

where 168 is the query ID (not the process ID used to cancel the query).

Alternatively, you can specify a custom confirmation message to display instead of the default message. To specify a custom message, include your message in single quotation marks at the end of the CANCEL command:

```
cancel 802 'Long-running query';
```

The session where the query was running displays the following message:

```
ERROR: Long-running query
```

## CLOSE

(Optional) Closes all of the free resources that are associated with an open cursor. [COMMIT \(p. 676\)](#), [END \(p. 879\)](#), and [ROLLBACK \(p. 927\)](#) automatically close the cursor, so it isn't necessary to use the CLOSE command to explicitly close the cursor.

For more information, see [DECLARE \(p. 855\)](#), [FETCH \(p. 886\)](#).

### Syntax

```
CLOSE cursor
```

### Parameters

*cursor*

Name of the cursor to close.

## CLOSE example

The following commands close the cursor and perform a commit, which ends the transaction:

```
close movie_cursor;
commit;
```

## COMMENT

Creates or changes a comment about a database object.

### Syntax

```
COMMENT ON
{
TABLE object_name |
```

```
COLUMN object_name.column_name |
CONSTRAINT constraint_name ON table_name |
DATABASE object_name |
VIEW object_name
}
IS 'text' | NULL
```

## Parameters

*object\_name*

Name of the database object being commented on. You can add a comment to the following objects:

- TABLE
- COLUMN (also takes a *column\_name*).
- CONSTRAINT (also takes a *constraint\_name* and *table\_name*).
- DATABASE
- VIEW

IS '*text*' | NULL

The comment text that you want to add or replace for the specified object. The *text* string is data type TEXT. Enclose the comment in single quotation marks. Set the value to NULL to remove the comment text.

*column\_name*

Name of the column being commented on. Parameter of COLUMN. Follows a table specified in *object\_name*.

*constraint\_name*

Name of the constraint that is being commented on. Parameter of CONSTRAINT.

*table\_name*

Name of a table containing the constraint. Parameter of CONSTRAINT.

## Usage notes

You must be a superuser or the owner of a database object to add or update a comment.

Comments on databases may only be applied to the current database. A warning message is displayed if you attempt to comment on a different database. The same warning is displayed for comments on databases that don't exist.

## Examples

The following example adds a comment to the SALES table.

```
COMMENT ON TABLE sales IS 'This table stores tickets sales data';
```

The following example displays the comment on the SALES table.

```
select obj_description('public.sales'::regclass);
obj_description

This table stores tickets sales data
```

The following example removes a comment from the SALES table.

```
COMMENT ON TABLE sales IS NULL;
```

The following example adds a comment to the EVENTID column of the SALES table.

```
COMMENT ON COLUMN sales.eventid IS 'Foreign-key reference to the EVENT table.';
```

The following example displays a comment on the EVENTID column (column number 5) of the SALES table.

```
select col_description('public.sales'::regclass, 5::integer);

col_description

Foreign-key reference to the EVENT table.
```

The following example adds a descriptive comment to the EVENT table.

```
comment on table event is 'Contains listings of individual events.';
```

To view comments, query the PG\_DESCRIPTION system catalog. The following example returns the description for the EVENT table.

```
select * from pg_catalog.pg_description
where objoid =
(select oid from pg_class where relname = 'event'
and relnamespace =
(select oid from pg_catalog.pg_namespace where nspname = 'public'));

objoid | classoid | objsubid | description
-----+-----+-----+-----
116658 | 1259 | 0 | Contains listings of individual events.
```

## COMMIT

Commits the current transaction to the database. This command makes the database updates from the transaction permanent.

### Syntax

```
COMMIT [WORK | TRANSACTION]
```

### Parameters

#### WORK

Optional keyword. This keyword isn't supported within a stored procedure.

#### TRANSACTION

Optional keyword. WORK and TRANSACTION are synonyms. Neither is supported within a stored procedure.

For information about using COMMIT within a stored procedure, see [Managing transactions \(p. 169\)](#).

## Examples

Each of the following examples commits the current transaction to the database:

```
commit;
```

```
commit work;
```

```
commit transaction;
```

## COPY

Loads data into a table from data files or from an Amazon DynamoDB table. The files can be located in an Amazon Simple Storage Service (Amazon S3) bucket, an Amazon EMR cluster, or a remote host that is accessed using a Secure Shell (SSH) connection.

**Note**

Amazon Redshift Spectrum external tables are read-only. You can't COPY to an external table.

The COPY command appends the input data as additional rows to the table.

The maximum size of a single input row from any source is 4 MB.

**Note**

To use the COPY command, you must have [INSERT \(p. 891\)](#) privilege for the Amazon Redshift table.

**Topics**

- [COPY syntax \(p. 677\)](#)
- [COPY syntax overview \(p. 677\)](#)
- [COPY JOB \(preview\) \(p. 681\)](#)
- [COPY parameter reference \(p. 683\)](#)
- [Usage notes \(p. 714\)](#)
- [COPY examples \(p. 726\)](#)

## COPY syntax

```
COPY table-name
[column-list]
FROM data-source
authorization
[[FORMAT] [AS] data-format]
[parameter [argument] [, ...]]
```

## COPY syntax overview

You can perform a COPY operation with as few as three parameters: a table name, a data source, and authorization to access the data.

Amazon Redshift extends the functionality of the COPY command to enable you to load data in several data formats from multiple data sources, control access to load data, manage data transformations, and manage the load operation.

This section presents the required COPY command parameters and groups the optional parameters by function. Subsequent topics describe each parameter and explain how various options work together. You can also go directly to a parameter description by using the alphabetical parameter list.

### Topics

- [Required parameters \(p. 678\)](#)
- [Optional parameters \(p. 679\)](#)
- [Using the COPY command \(p. 681\)](#)

## Required parameters

The COPY command requires three elements:

- [Table Name \(p. 678\)](#)
- [Data Source \(p. 678\)](#)
- [Authorization \(p. 679\)](#)

The simplest COPY command uses the following format.

```
COPY table-name
FROM data-source
authorization;
```

The following example creates a table named CATDEMO, and then loads the table with sample data from a data file in Amazon S3 named category\_pipe.txt.

```
create table catdemo(catid smallint, catgroup varchar(10), catname varchar(10), catdesc
varchar(50));
```

In the following example, the data source for the COPY command is a data file named category\_pipe.txt in the tickit folder of an Amazon S3 bucket named awssampledbuswest2. The COPY command is authorized to access the Amazon S3 bucket through an AWS Identity and Access Management (IAM) role. If your cluster has an existing IAM role with permission to access Amazon S3 attached, you can substitute your role's Amazon Resource Name (ARN) in the following COPY command and run it.

```
copy catdemo
from 's3://awssampledbuswest2/ticket/category_pipe.txt'
iam_role 'arn:aws:iam::<aws-account-id>:role/<role-name>'
region 'us-west-2';
```

For steps to create an IAM role, see [Step 2: Create an IAM Role](#) in the Amazon Redshift Getting Started Guide. For complete instructions on how to use COPY commands to load sample data, including instructions for loading data from other AWS regions, see [Step 6: Load Sample Data from Amazon S3](#) in the Amazon Redshift Getting Started Guide..

*table-name*

The name of the target table for the COPY command. The table must already exist in the database. The table can be temporary or persistent. The COPY command appends the new input data to any existing rows in the table.

FROM *data-source*

The location of the source data to be loaded into the target table. A manifest file can be specified with some data sources.

The most commonly used data repository is an Amazon S3 bucket. You can also load from data files located in an Amazon EMR cluster, an Amazon EC2 instance, or a remote host that your cluster can access using an SSH connection, or you can load directly from a DynamoDB table.

- [COPY from Amazon S3 \(p. 683\)](#)
- [COPY from Amazon EMR \(p. 687\)](#)
- [COPY from remote host \(SSH\) \(p. 688\)](#)
- [COPY from Amazon DynamoDB \(p. 691\)](#)

#### Authorization

A clause that indicates the method that your cluster uses for authentication and authorization to access other AWS resources. The COPY command needs authorization to access data in another AWS resource, including in Amazon S3, Amazon EMR, Amazon DynamoDB, and Amazon EC2. You can provide that authorization by referencing an IAM role that is attached to your cluster or by providing the access key ID and secret access key for an IAM user.

- [Authorization parameters \(p. 693\)](#)
- [Role-based access control \(p. 715\)](#)
- [Key-based access control \(p. 716\)](#)

## Optional parameters

You can optionally specify how COPY maps field data to columns in the target table, define source data attributes to enable the COPY command to correctly read and parse the source data, and manage which operations the COPY command performs during the load process.

- [Column mapping options \(p. 695\)](#)
- [Data format parameters \(p. 679\)](#)
- [Data conversion parameters \(p. 680\)](#)
- [Data load operations \(p. 680\)](#)

### Column mapping

By default, COPY inserts field values into the target table's columns in the same order as the fields occur in the data files. If the default column order will not work, you can specify a column list or use JSONPath expressions to map source data fields to the target columns.

- [Column List \(p. 696\)](#)
- [JSONPaths File \(p. 696\)](#)

### Data format parameters

You can load data from text files in fixed-width, character-delimited, comma-separated values (CSV), or JSON format, or from Avro files.

By default, the COPY command expects the source data to be in character-delimited UTF-8 text files. The default delimiter is a pipe character ( | ). If the source data is in another format, use the following parameters to specify the data format.

- [FORMAT \(p. 697\)](#)
- [CSV \(p. 697\)](#)
- [DELIMITER \(p. 697\)](#)
- [FIXEDWIDTH \(p. 697\)](#)

- [SHAPEFILE \(p. 698\)](#)
- [AVRO \(p. 698\)](#)
- [JSON \(p. 700\)](#)
- [ENCRYPTED \(p. 686\)](#)
- [BZIP2 \(p. 705\)](#)
- [GZIP \(p. 706\)](#)
- [LZOP \(p. 706\)](#)
- [PARQUET \(p. 705\)](#)
- [ORC \(p. 705\)](#)
- [ZSTD \(p. 706\)](#)

### Data conversion parameters

As it loads the table, COPY attempts to implicitly convert the strings in the source data to the data type of the target column. If you need to specify a conversion that is different from the default behavior, or if the default conversion results in errors, you can manage data conversions by specifying the following parameters.

- [ACCEPTANYDATE \(p. 706\)](#)
- [ACCEPTINVCHARS \(p. 707\)](#)
- [BLANKSASNULL \(p. 707\)](#)
- [DATEFORMAT \(p. 707\)](#)
- [EMPTYASNULL \(p. 707\)](#)
- [ENCODING \(p. 707\)](#)
- [ESCAPE \(p. 708\)](#)
- [EXPLICIT\\_IDS \(p. 709\)](#)
- [FILLRECORD \(p. 709\)](#)
- [IGNOREBLANKLINES \(p. 709\)](#)
- [IGNOREHEADER \(p. 710\)](#)
- [NULL AS \(p. 710\)](#)
- [REMOVEQUOTES \(p. 710\)](#)
- [ROUNDDEC \(p. 710\)](#)
- [TIMEFORMAT \(p. 710\)](#)
- [TRIMBLANKS \(p. 711\)](#)
- [TRUNCATECOLUMNS \(p. 711\)](#)

### Data load operations

Manage the default behavior of the load operation for troubleshooting or to reduce load times by specifying the following parameters.

- [COMPROWS \(p. 711\)](#)
- [COMPUPDATE \(p. 712\)](#)
- [IGNOREALLERRORS \(p. 712\)](#)
- [MAXERROR \(p. 712\)](#)
- [NOLOAD \(p. 712\)](#)
- [STATUPDATE \(p. 713\)](#)

## Using the COPY command

For more information about how to use the COPY command, see the following topics:

- [COPY examples \(p. 726\)](#)
- [Usage notes \(p. 714\)](#)
- [Tutorial: Loading data from Amazon S3 \(p. 124\)](#)
- [Amazon Redshift best practices for loading data \(p. 17\)](#)
- [Using a COPY command to load data \(p. 63\)](#)
  - [Loading data from Amazon S3 \(p. 66\)](#)
  - [Loading data from Amazon EMR \(p. 74\)](#)
  - [Loading data from remote hosts \(p. 78\)](#)
  - [Loading data from an Amazon DynamoDB table \(p. 84\)](#)
- [Troubleshooting data loads \(p. 90\)](#)

## COPY JOB (preview)

*This is prerelease documentation for auto-copy (SQL COPY JOB), which is in preview release. The documentation and the feature are both subject to change. We recommend that you use this feature only in test environments, and not in production environments. Public preview will end on May 1, 2023. Preview clusters and preview serverless workgroups and namespaces will be removed automatically two weeks after the end of the preview. For preview terms and conditions, see [Betas and Previews in AWS Service Terms](#).*

For information about using this command in preview, see [Continuous file ingestion from Amazon S3 \(preview\) \(p. 94\)](#).

Manages COPY commands that load data into a table. The COPY JOB command is an extension of the COPY command and automates data loading from Amazon S3 buckets. When you create a COPY job, Amazon Redshift detects when new Amazon S3 files are created in a specified path, and then loads them automatically without your intervention. The same parameters that are used in the original COPY command are used when loading the data. Amazon Redshift keeps track of the loaded files to verify that they are loaded only one time.

### Note

For information about the COPY command, including usage, parameters, and permissions, see [COPY \(p. 677\)](#).

### Required permission

To run the COPY command of a COPY JOB, you must have INSERT privilege of the table being loaded.

The IAM role specified with the COPY command must have permission to access the data to load. For more information, see [IAM permissions for COPY, UNLOAD, and CREATE LIBRARY \(p. 718\)](#).

### Syntax

Create a copy job. The parameters of the COPY command are saved with the copy job.

```
COPY copy-command JOB CREATE job-name
[AUTO ON | OFF]
```

Change the configuration of a copy job.

```
COPY JOB ALTER job-name
[AUTO ON | OFF]
```

Run a copy job. The stored COPY command parameters are used.

```
COPY JOB RUN job-name
```

List all copy jobs.

```
COPY JOB LIST
```

Show the details of a copy job.

```
COPY JOB SHOW job-name
```

Delete a copy job.

```
COPY JOB DROP job-name
```

## Parameters

*copy-command*

A COPY command that loads data from Amazon S3 to Amazon Redshift. The clause contains COPY parameters that define the Amazon S3 bucket, target table, IAM role, and other parameters used when loading data. All COPY command parameters for an Amazon S3 data load are supported except:

- The COPY JOB must be created using a COPY command that points to an empty Amazon S3 folder (it doesn't contain any files).
- You cannot specify a COPY command with the MAXERROR or IGNOREALLERRORS options.
- You cannot specify a manifest file. COPY JOB requires a designated Amazon S3 location to monitor for newly created files.
- You cannot specify an Amazon S3 file with the following columnar data formats: ORC and Parquet.
- You cannot specify a COPY command with authorization types like Access and Secret keys. Only COPY commands that use the IAM\_ROLE parameter for authorization are supported. For more information, see [Authorization parameters \(p. 693\)](#).
- The COPY JOB doesn't support the default IAM role associated with the cluster. You must specify the IAM\_ROLE in the COPY command.

For more information, see [COPY from Amazon S3 \(p. 683\)](#).

*job-name*

The name of the job used to reference the COPY job.

[AUTO ON | OFF]

Clause that indicates whether Amazon S3 data is automatically loaded into Amazon Redshift tables.

- When ON, Amazon Redshift monitors the source Amazon S3 path for newly created files, and if found, a COPY command is run with the COPY parameters in the job definition. This is the default.

- When OFF, Amazon Redshift does not run the COPY JOB automatically.

## Usage notes

The options of the COPY command aren't validated until run time. For example, an invalid IAM\_ROLE or an Amazon S3 data source results in runtime errors when the COPY JOB starts.

If the cluster is paused, COPY JOBS are not run.

To query COPY command files loaded and load errors, see [STL\\_LOAD\\_COMMITS \(p. 1523\)](#), [STL\\_LOAD\\_ERRORS \(p. 1526\)](#), [STL\\_LOADERROR\\_DETAIL \(p. 1528\)](#). For more information, see [Verifying that the data loaded correctly \(p. 86\)](#).

## Examples

The following example shows creating a COPY JOB to load data from an Amazon S3 bucket.

```
COPY public.target_table
FROM 's3://mybucket-bucket/staging-folder'
IAM_ROLE 'arn:aws:iam::123456789012:role/MyLoadRoleName'
JOB CREATE my_copy_job_name
AUTO ON;
```

## COPY parameter reference

### Topics

- [Data sources \(p. 683\)](#)
- [Authorization parameters \(p. 693\)](#)
- [Column mapping options \(p. 695\)](#)
- [Data format parameters \(p. 696\)](#)
- [Data load operations \(p. 711\)](#)
- [Alphabetical parameter list \(p. 713\)](#)

## Data sources

You can load data from text files in an Amazon S3 bucket, in an Amazon EMR cluster, or on a remote host that your cluster can access using an SSH connection. You can also load data directly from a DynamoDB table.

The maximum size of a single input row from any source is 4 MB.

To export data from a table to a set of files in an Amazon S3, use the [UNLOAD \(p. 991\)](#) command.

### Topics

- [COPY from Amazon S3 \(p. 683\)](#)
- [COPY from Amazon EMR \(p. 687\)](#)
- [COPY from remote host \(SSH\) \(p. 688\)](#)
- [COPY from Amazon DynamoDB \(p. 691\)](#)

## COPY from Amazon S3

To load data from files located in one or more S3 buckets, use the FROM clause to indicate how COPY locates the files in Amazon S3. You can provide the object path to the data files as part of the FROM

clause, or you can provide the location of a manifest file that contains a list of Amazon S3 object paths. COPY from Amazon S3 uses an HTTPS connection. Ensure that the S3 IP ranges are added to your allow list. To learn more about the required S3 IP ranges, see [Network isolation](#).

**Important**

If the Amazon S3 buckets that hold the data files don't reside in the same AWS Region as your cluster, you must use the [REGION \(p. 686\)](#) parameter to specify the Region in which the data is located.

**Topics**

- [Syntax \(p. 684\)](#)
- [Examples \(p. 684\)](#)
- [Optional parameters \(p. 687\)](#)
- [Unsupported parameters \(p. 687\)](#)

**Syntax**

```
FROM { 's3://objectpath' | 's3://manifest_file' }
authorization
| MANIFEST
| ENCRYPTED
| REGION [AS] 'aws-region'
| optional-parameters
```

**Examples**

The following example uses an object path to load data from Amazon S3.

```
copy customer
from 's3://mybucket/customer'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

The following example uses a manifest file to load data from Amazon S3.

```
copy customer
from 's3://mybucket/cust.manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

**Parameters**

**FROM**

The source of the data to be loaded. For more information about the encoding of the Amazon S3 file, see [Data conversion parameters \(p. 706\)](#).

's3://copy\_from\_s3\_objectpath'

Specifies the path to the Amazon S3 objects that contain the data—for example, 's3://mybucket/custdata.txt'. The `s3://copy_from_s3_objectpath` parameter can reference a single file or a set of objects or folders that have the same key prefix. For example, the name `custdata.txt` is a key prefix that refers to a number of physical files: `custdata.txt,custdata.txt.1,custdata.txt.2,custdata.txt.bak`, and so on. The key prefix can also reference a number of folders. For example, 's3://mybucket/custfolder' refers to the folders `custfolder,custfolder_1,custfolder_2`, and so on. If a key prefix references multiple folders, all of the files in the folders are loaded. If a key prefix matches a file as

well as a folder, such as `custfolder.log`, COPY attempts to load the file also. If a key prefix might result in COPY attempting to load unwanted files, use a manifest file. For more information, see [copy\\_from\\_s3\\_manifest\\_file \(p. 685\)](#), following.

**Important**

If the S3 bucket that holds the data files doesn't reside in the same AWS Region as your cluster, you must use the [REGION \(p. 686\)](#) parameter to specify the Region in which the data is located.

For more information, see [Loading data from Amazon S3 \(p. 66\)](#).

'`s3://copy_from_s3_manifest_file'`

Specifies the Amazon S3 object key for a manifest file that lists the data files to be loaded. The '`s3://copy_from_s3_manifest_file`' argument must explicitly reference a single file—for example, '`s3://mybucket/manifest.txt`'. It can't reference a key prefix.

The manifest is a text file in JSON format that lists the URL of each file that is to be loaded from Amazon S3. The URL includes the bucket name and full object path for the file. The files that are specified in the manifest can be in different buckets, but all the buckets must be in the same AWS Region as the Amazon Redshift cluster. If a file is listed twice, the file is loaded twice. The following example shows the JSON for a manifest that loads three files.

```
{
 "entries": [
 {"url": "s3://mybucket-alpha/custdata.1", "mandatory": true},
 {"url": "s3://mybucket-alpha/custdata.2", "mandatory": true},
 {"url": "s3://mybucket-beta/custdata.1", "mandatory": false}
]
}
```

The double quotation mark characters are required, and must be simple quotation marks (0x22), not slanted or "smart" quotation marks. Each entry in the manifest can optionally include a mandatory flag. If mandatory is set to true, COPY terminates if it doesn't find the file for that entry; otherwise, COPY will continue. The default value for mandatory is false.

When loading from data files in ORC or Parquet format, a meta field is required, as shown in the following example.

```
{
 "entries": [
 {
 "url": "s3://mybucket-alpha/orc/2013-10-04-custdata",
 "mandatory": true,
 "meta": {
 "content_length": 99
 }
 },
 {
 "url": "s3://mybucket-beta/orc/2013-10-05-custdata",
 "mandatory": true,
 "meta": {
 "content_length": 99
 }
 }
]
}
```

The manifest file must not be encrypted or compressed, even if the ENCRYPTED, GZIP, LZOP, BZIP2, or ZSTD options are specified. COPY returns an error if the specified manifest file isn't found or the manifest file isn't properly formed.

If a manifest file is used, the MANIFEST parameter must be specified with the COPY command. If the MANIFEST parameter isn't specified, COPY assumes that the file specified with FROM is a data file.

For more information, see [Loading data from Amazon S3 \(p. 66\)](#).

#### *authorization*

The COPY command needs authorization to access data in another AWS resource, including in Amazon S3, Amazon EMR, Amazon DynamoDB, and Amazon EC2. You can provide that authorization by referencing an AWS Identity and Access Management (IAM) role that is attached to your cluster (role-based access control) or by providing the access credentials for a user (key-based access control). For increased security and flexibility, we recommend using IAM role-based access control. For more information, see [Authorization parameters \(p. 693\)](#).

#### MANIFEST

Specifies that a manifest is used to identify the data files to be loaded from Amazon S3. If the MANIFEST parameter is used, COPY loads data from the files listed in the manifest referenced by 's3://copy\_from\_s3\_manifest\_file'. If the manifest file isn't found, or isn't properly formed, COPY fails. For more information, see [Using a manifest to specify data files \(p. 71\)](#).

#### ENCRYPTED

A clause that specifies that the input files on Amazon S3 are encrypted using client-side encryption with customer managed keys. For more information, see [Loading encrypted data files from Amazon S3 \(p. 73\)](#). Don't specify ENCRYPTED if the input files are encrypted using Amazon S3 server-side encryption (SSE-KMS or SSE-S3). COPY reads server-side encrypted files automatically.

If you specify the ENCRYPTED parameter, you must also specify the [MASTER\\_SYMMETRIC\\_KEY \(p. 686\)](#) parameter or include the `master_symmetric_key` value in the [CREDENTIALS \(p. 694\)](#) string.

If the encrypted files are in compressed format, add the GZIP, LZOP, BZIP2, or ZSTD parameter.

Manifest files and JSONPaths files must not be encrypted, even if the ENCRYPTED option is specified.

#### MASTER\_SYMMETRIC\_KEY 'root\_key'

The root symmetric key that was used to encrypt data files on Amazon S3. If MASTER\_SYMMETRIC\_KEY is specified, the [ENCRYPTED \(p. 686\)](#) parameter must also be specified. MASTER\_SYMMETRIC\_KEY can't be used with the CREDENTIALS parameter. For more information, see [Loading encrypted data files from Amazon S3 \(p. 73\)](#).

If the encrypted files are in compressed format, add the GZIP, LZOP, BZIP2, or ZSTD parameter.

#### REGION [AS] 'aws-region'

Specifies the AWS Region where the source data is located. REGION is required for COPY from an Amazon S3 bucket or an DynamoDB table when the AWS resource that contains the data isn't in the same Region as the Amazon Redshift cluster.

The value for `aws_region` must match a Region listed in the [Amazon Redshift regions and endpoints](#) table.

If the REGION parameter is specified, all resources, including a manifest file or multiple Amazon S3 buckets, must be located in the specified Region.

#### Note

Transferring data across Regions incurs additional charges against the Amazon S3 bucket or the DynamoDB table that contains the data. For more information about pricing, see [Data Transfer OUT From Amazon S3 To Another AWS Region](#) on the [Amazon S3 Pricing](#) page and [Data Transfer OUT](#) on the [Amazon DynamoDB Pricing](#) page.

By default, COPY assumes that the data is located in the same Region as the Amazon Redshift cluster.

### Optional parameters

You can optionally specify the following parameters with COPY from Amazon S3:

- [Column mapping options \(p. 695\)](#)
- [Data format parameters \(p. 697\)](#)
- [Data conversion parameters \(p. 706\)](#)
- [Data load operations \(p. 711\)](#)

### Unsupported parameters

You can't use the following parameters with COPY from Amazon S3:

- SSH
- READRATIO

### COPY from Amazon EMR

You can use the COPY command to load data in parallel from an Amazon EMR cluster configured to write text files to the cluster's Hadoop Distributed File System (HDFS) in the form of fixed-width files, character-delimited files, CSV files, JSON-formatted files, or Avro files.

#### Topics

- [Syntax \(p. 687\)](#)
- [Example \(p. 687\)](#)
- [Parameters \(p. 687\)](#)
- [Supported parameters \(p. 688\)](#)
- [Unsupported parameters \(p. 688\)](#)

### Syntax

```
FROM 'emr://emr_cluster_id/hdfs_filepath'
authorization
[optional_parameters]
```

### Example

The following example loads data from an Amazon EMR cluster.

```
copy sales
from 'emr://j-SAMPLE2B500FC/myoutput/part-*'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

### Parameters

#### FROM

The source of the data to be loaded.

'emr://emr\_cluster\_id/hdfs\_file\_path'

The unique identifier for the Amazon EMR cluster and the HDFS file path that references the data files for the COPY command. The HDFS data file names must not contain the wildcard characters asterisk (\*) and question mark (?).

**Note**

The Amazon EMR cluster must continue running until the COPY operation completes. If any of the HDFS data files are changed or deleted before the COPY operation completes, you might have unexpected results, or the COPY operation might fail.

You can use the wildcard characters asterisk (\*) and question mark (?) as part of the *hdfs\_file\_path* argument to specify multiple files to be loaded. For example, 'emr://j-SAMPLE2B500FC/myoutput/part\*' identifies the files part-0000, part-0001, and so on. If the file path doesn't contain wildcard characters, it is treated as a string literal. If you specify only a folder name, COPY attempts to load all files in the folder.

**Important**

If you use wildcard characters or use only the folder name, verify that no unwanted files will be loaded. For example, some processes might write a log file to the output folder.

For more information, see [Loading data from Amazon EMR \(p. 74\)](#).

*authorization*

The COPY command needs authorization to access data in another AWS resource, including in Amazon S3, Amazon EMR, Amazon DynamoDB, and Amazon EC2. You can provide that authorization by referencing an AWS Identity and Access Management (IAM) role that is attached to your cluster (role-based access control) or by providing the access credentials for a user (key-based access control). For increased security and flexibility, we recommend using IAM role-based access control.

For more information, see [Authorization parameters \(p. 693\)](#).

## Supported parameters

You can optionally specify the following parameters with COPY from Amazon EMR:

- [Column mapping options \(p. 695\)](#)
- [Data format parameters \(p. 697\)](#)
- [Data conversion parameters \(p. 706\)](#)
- [Data load operations \(p. 711\)](#)

## Unsupported parameters

You can't use the following parameters with COPY from Amazon EMR:

- ENCRYPTED
- MANIFEST
- REGION
- READRATIO
- SSH

## COPY from remote host (SSH)

You can use the COPY command to load data in parallel from one or more remote hosts, such Amazon Elastic Compute Cloud (Amazon EC2) instances or other computers. COPY connects to the remote hosts using Secure Shell (SSH) and runs commands on the remote hosts to generate text output. The remote host can be an EC2 Linux instance or another Unix or Linux computer configured to accept SSH

connections. Amazon Redshift can connect to multiple hosts, and can open multiple SSH connections to each host. Amazon Redshift sends a unique command through each connection to generate text output to the host's standard output, which Amazon Redshift then reads as it does a text file.

Use the FROM clause to specify the Amazon S3 object key for the manifest file that provides the information COPY uses to open SSH connections and run the remote commands.

### Topics

- [Syntax \(p. 689\)](#)
- [Examples \(p. 689\)](#)
- [Parameters \(p. 689\)](#)
- [Optional parameters \(p. 691\)](#)
- [Unsupported parameters \(p. 691\)](#)

### Important

If the S3 bucket that holds the manifest file doesn't reside in the same AWS Region as your cluster, you must use the REGION parameter to specify the Region in which the bucket is located.

### Syntax

```
FROM 's3://ssh_manifest_file' }
authorization
SSH
| optional-parameters
```

### Examples

The following example uses a manifest file to load data from a remote host using SSH.

```
copy sales
from 's3://mybucket/ssh_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
ssh;
```

### Parameters

#### FROM

The source of the data to be loaded.

's3://copy\_from\_ssh\_manifest\_file'

The COPY command can connect to multiple hosts using SSH, and can create multiple SSH connections to each host. COPY runs a command through each host connection, and then loads the output from the commands in parallel into the table. The s3://copy\_from\_ssh\_manifest\_file argument specifies the Amazon S3 object key for the manifest file that provides the information COPY uses to open SSH connections and run the remote commands.

The s3://copy\_from\_ssh\_manifest\_file argument must explicitly reference a single file; it can't be a key prefix. The following shows an example:

```
's3://mybucket/ssh_manifest.txt'
```

The manifest file is a text file in JSON format that Amazon Redshift uses to connect to the host. The manifest file specifies the SSH host endpoints and the commands that will be run on the hosts

to return data to Amazon Redshift. Optionally, you can include the host public key, the login user name, and a mandatory flag for each entry. The following example shows a manifest file that creates two SSH connections:

```
{
 "entries": [
 {"endpoint": "<ssh_endpoint_or_IP>",
 "command": "<remote_command>",
 "mandatory": true,
 "publickey": "<public_key>",
 "username": "<host_user_name>"},
 {"endpoint": "<ssh_endpoint_or_IP>",
 "command": "<remote_command>",
 "mandatory": true,
 "publickey": "<public_key>",
 "username": "<host_user_name>"}
]
}
```

The manifest file contains one "entries" construct for each SSH connection. You can have multiple connections to a single host or multiple connections to multiple hosts. The double quotation mark characters are required as shown, both for the field names and the values. The quotation mark characters must be simple quotation marks (0x22), not slanted or "smart" quotation marks. The only value that doesn't need double quotation mark characters is the Boolean value `true` or `false` for the "mandatory" field.

The following list describes the fields in the manifest file.

#### endpoint

The URL address or IP address of the host—for example, "ec2-111-222-333.compute-1.amazonaws.com", or "198.51.100.0".

#### command

The command to be run by the host to generate text output or binary output in gzip, lzop, bzip2, or zstd format. The command can be any command that the user "`host_user_name`" has permission to run. The command can be as simple as printing a file, or it can query a database or launch a script. The output (text file, gzip binary file, lzop binary file, or bzip2 binary file) must be in a form that the Amazon Redshift COPY command can ingest. For more information, see [Preparing your input data \(p. 65\)](#).

#### publickey

(Optional) The public key of the host. If provided, Amazon Redshift will use the public key to identify the host. If the public key isn't provided, Amazon Redshift will not attempt host identification. For example, if the remote host's public key is `ssh-rsa AbcCbaxxx...Example root@amazon.com`, type the following text in the public key field: "AbcCbaxxx...Example"

#### mandatory

(Optional) A clause that indicates whether the COPY command should fail if the connection attempt fails. The default is `false`. If Amazon Redshift doesn't successfully make at least one connection, the COPY command fails.

#### username

(Optional) The user name that will be used to log on to the host system and run the remote command. The user login name must be the same as the login that was used to add the Amazon Redshift cluster's public key to the host's authorized keys file. The default username is `redshift`.

For more information about creating a manifest file, see [Loading data process \(p. 79\)](#).

To COPY from a remote host, the SSH parameter must be specified with the COPY command. If the SSH parameter isn't specified, COPY assumes that the file specified with FROM is a data file and will fail.

If you use automatic compression, the COPY command performs two data read operations, which means it will run the remote command twice. The first read operation is to provide a data sample for compression analysis, then the second read operation actually loads the data. If executing the remote command twice might cause a problem, you should disable automatic compression. To disable automatic compression, run the COPY command with the COMPUPDATE parameter set to OFF. For more information, see [Loading tables with automatic compression \(p. 87\)](#).

For detailed procedures for using COPY from SSH, see [Loading data from remote hosts \(p. 78\)](#).

#### *authorization*

The COPY command needs authorization to access data in another AWS resource, including in Amazon S3, Amazon EMR, Amazon DynamoDB, and Amazon EC2. You can provide that authorization by referencing an AWS Identity and Access Management (IAM) role that is attached to your cluster (role-based access control) or by providing the access credentials for a user (key-based access control). For increased security and flexibility, we recommend using IAM role-based access control. For more information, see [Authorization parameters \(p. 693\)](#).

#### SSH

A clause that specifies that data is to be loaded from a remote host using the SSH protocol. If you specify SSH, you must also provide a manifest file using the [`s3://copy\_from\_ssh\_manifest\_file \(p. 689\)`](#) argument.

##### **Note**

If you are using SSH to copy from a host using a private IP address in a remote VPC, the VPC must have enhanced VPC routing enabled. For more information about Enhanced VPC routing, see [Amazon Redshift Enhanced VPC Routing](#).

#### Optional parameters

You can optionally specify the following parameters with COPY from SSH:

- [Column mapping options \(p. 695\)](#)
- [Data format parameters \(p. 697\)](#)
- [Data conversion parameters \(p. 706\)](#)
- [Data load operations \(p. 711\)](#)

#### Unsupported parameters

You can't use the following parameters with COPY from SSH:

- ENCRYPTED
- MANIFEST
- READRATIO

#### [COPY from Amazon DynamoDB](#)

To load data from an existing DynamoDB table, use the FROM clause to specify the DynamoDB table name.

#### Topics

- [Syntax \(p. 692\)](#)

- [Examples \(p. 692\)](#)
- [Optional parameters \(p. 693\)](#)
- [Unsupported parameters \(p. 693\)](#)

**Important**

If the DynamoDB table doesn't reside in the same region as your Amazon Redshift cluster, you must use the REGION parameter to specify the region in which the data is located.

**Syntax**

```
FROM 'dynamodb://table-name'
authorization
READRATIO ratio
| REGION [AS] 'aws_region'
| optional-parameters
```

**Examples**

The following example loads data from a DynamoDB table.

```
copy favoritemovies from 'dynamodb://ProductCatalog'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
readratio 50;
```

**Parameters**

**FROM**

The source of the data to be loaded.

'dynamodb://table-name'

The name of the DynamoDB table that contains the data, for example 'dynamodb://ProductCatalog'. For details about how DynamoDB attributes are mapped to Amazon Redshift columns, see [Loading data from an Amazon DynamoDB table \(p. 84\)](#).

A DynamoDB table name is unique to an AWS account, which is identified by the AWS access credentials.

*authorization*

The COPY command needs authorization to access data in another AWS resource, including in Amazon S3, Amazon EMR, DynamoDB, and Amazon EC2. You can provide that authorization by referencing an AWS Identity and Access Management (IAM) role that is attached to your cluster (role-based access control) or by providing the access credentials for a user (key-based access control). For increased security and flexibility, we recommend using IAM role-based access control. For more information, see [Authorization parameters \(p. 693\)](#).

**READRATIO [AS] *ratio***

The percentage of the DynamoDB table's provisioned throughput to use for the data load.

READRATIO is required for COPY from DynamoDB. It can't be used with COPY from Amazon S3. We highly recommend setting the ratio to a value less than the average unused provisioned throughput. Valid values are integers 1–200.

**Important**

Setting READRATIO to 100 or higher enables Amazon Redshift to consume the entirety of the DynamoDB table's provisioned throughput, which seriously degrades the performance of concurrent read operations against the same table during the COPY session. Write

traffic is unaffected. Values higher than 100 are allowed to troubleshoot rare scenarios when Amazon Redshift fails to fulfill the provisioned throughput of the table. If you load data from DynamoDB to Amazon Redshift on an ongoing basis, consider organizing your DynamoDB tables as a time series to separate live traffic from the COPY operation.

## Optional parameters

You can optionally specify the following parameters with COPY from Amazon DynamoDB:

- [Column mapping options \(p. 695\)](#)
- The following data conversion parameters are supported:
  - [ACCEPTANYDATE \(p. 706\)](#)
  - [BLANKSASNLL \(p. 707\)](#)
  - [DATEFORMAT \(p. 707\)](#)
  - [EMPTYASNLL \(p. 707\)](#)
  - [ROUNDDEC \(p. 710\)](#)
  - [TIMEFORMAT \(p. 710\)](#)
  - [TRIMBLANKS \(p. 711\)](#)
  - [TRUNCATECOLUMNS \(p. 711\)](#)
- [Data load operations \(p. 711\)](#)

## Unsupported parameters

You can't use the following parameters with COPY from DynamoDB:

- All data format parameters
- ESCAPE
- FILLRECORD
- IGNOREBLANKLINES
- IGNOREHEADER
- NULL
- REMOVEQUOTES
- ACCEPTINVCHARS
- MANIFEST
- ENCRYPTED

## Authorization parameters

The COPY command needs authorization to access data in another AWS resource, including in Amazon S3, Amazon EMR, Amazon DynamoDB, and Amazon EC2. You can provide that authorization by referencing an [AWS Identity and Access Management \(IAM\) role](#) that is attached to your cluster (*role-based access control*). You can encrypt your load data on Amazon S3.

The following topics provide more details and examples of authentication options:

- [IAM permissions for COPY, UNLOAD, and CREATE LIBRARY \(p. 718\)](#)
- [Role-based access control \(p. 715\)](#)
- [Key-based access control \(p. 716\)](#)

Use one of the following to provide authorization for the COPY command:

- [IAM\\_ROLE \(p. 694\)](#) parameter
- [ACCESS\\_KEY\\_ID and SECRET\\_ACCESS\\_KEY \(p. 694\)](#) parameters
- [CREDENTIALS \(p. 694\)](#) clause

```
IAM_ROLE { default | 'arn:aws:iam::<AWS account-id>:role/<role-name>' }
```

Use the default keyword to have Amazon Redshift use the IAM role that is set as default and associated with the cluster when the COPY command runs.

Use the Amazon Resource Name (ARN) for an IAM role that your cluster uses for authentication and authorization. If you specify IAM\_ROLE, you can't use ACCESS\_KEY\_ID and SECRET\_ACCESS\_KEY, SESSION\_TOKEN, or CREDENTIALS.

The following shows the syntax for the IAM\_ROLE parameter.

```
IAM_ROLE { default | 'arn:aws:iam::<AWS account-id>:role/<role-name>' }
```

For more information, see [Role-based access control \(p. 715\)](#).

ACCESS\_KEY\_ID '*access-key-id*' SECRET\_ACCESS\_KEY '*secret-access-key*'

This authorization method is not recommended.

**Note**

Instead of providing access credentials as plain text, we strongly recommend using role-based authentication by specifying the IAM\_ROLE parameter. For more information, see [Role-based access control \(p. 715\)](#).

SESSION\_TOKEN '*temporary-token*'

The session token for use with temporary access credentials. When SESSION\_TOKEN is specified, you must also use ACCESS\_KEY\_ID and SECRET\_ACCESS\_KEY to provide temporary access key credentials. If you specify SESSION\_TOKEN you can't use IAM\_ROLE or CREDENTIALS. For more information, see [Temporary security credentials \(p. 716\)](#) in the IAM User Guide.

**Note**

Instead of creating temporary security credentials, we strongly recommend using role-based authentication. When you authorize using an IAM role, Amazon Redshift automatically creates temporary user credentials for each session. For more information, see [Role-based access control \(p. 715\)](#).

The following shows the syntax for the SESSION\_TOKEN parameter with the ACCESS\_KEY\_ID and SECRET\_ACCESS\_KEY parameters.

```
ACCESS_KEY_ID '<access-key-id>'
SECRET_ACCESS_KEY '<secret-access-key>'
SESSION_TOKEN '<temporary-token>';
```

If you specify SESSION\_TOKEN you can't use CREDENTIALS or IAM\_ROLE.

[WITH] CREDENTIALS [AS] '*credentials-args*'

A clause that indicates the method your cluster will use when accessing other AWS resources that contain data files or manifest files. You can't use the CREDENTIALS parameter with IAM\_ROLE or ACCESS\_KEY\_ID and SECRET\_ACCESS\_KEY.

**Note**

For increased flexibility, we recommend using the [IAM\\_ROLE \(p. 694\)](#) parameter instead of the CREDENTIALS parameter.

Optionally, if the [ENCRYPTED \(p. 686\)](#) parameter is used, the *credentials-args* string also provides the encryption key.

The *credentials-args* string is case-sensitive and must not contain spaces.

The keywords WITH and AS are optional and are ignored.

You can specify either [role-based access control \(p. 715\)](#) or [key-based access control \(p. 716\)](#). In either case, the IAM role or user must have the permissions required to access the specified AWS resources. For more information, see [IAM permissions for COPY, UNLOAD, and CREATE LIBRARY \(p. 718\)](#).

**Note**

To safeguard your AWS credentials and protect sensitive data, we strongly recommend using role-based access control.

To specify role-based access control, provide the *credentials-args* string in the following format.

```
'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
```

To use temporary token credentials, you must provide the temporary access key ID, the temporary secret access key, and the temporary token. The *credentials-args* string is in the following format.

```
CREDENTIALS
'aws_access_key_id=<temporary-access-key-id>;aws_secret_access_key=<temporary-secret-access-key>;token=<temporary-token>'
```

For more information, see [Temporary security credentials \(p. 716\)](#).

If the [ENCRYPTED \(p. 686\)](#) parameter is used, the *credentials-args* string is in the following format, where *<root-key>* is the value of the root key that was used to encrypt the files.

```
CREDENTIALS
'<credentials-args>;master_symmetric_key=<root-key>'
```

For example, the following COPY command uses role-based access control with an encryption key.

```
copy customer from 's3://mybucket/mydata'
credentials
'aws_iam_role=arn:aws:iam::<account-id>:role/<role-name>;master_symmetric_key=<root-key>'
```

The following COPY command shows role-based access control with an encryption key.

```
copy customer from 's3://mybucket/mydata'
credentials
'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>;master_symmetric_key=<root-key>'
```

## Column mapping options

By default, COPY inserts values into the target table's columns in the same order as fields occur in the data files. If the default column order will not work, you can specify a column list or use JSONPath expressions to map source data fields to the target columns.

- [Column List \(p. 696\)](#)

- [JSONPaths File \(p. 696\)](#)

### Column list

You can specify a comma-separated list of column names to load source data fields into specific target columns. The columns can be in any order in the COPY statement, but when loading from flat files, such as in an Amazon S3 bucket, their order must match the order of the source data.

When loading from an Amazon DynamoDB table, order doesn't matter. The COPY command matches attribute names in the items retrieved from the DynamoDB table to column names in the Amazon Redshift table. For more information, see [Loading data from an Amazon DynamoDB table \(p. 84\)](#)

The format for a column list is as follows.

```
COPY tablename (column1 [,column2, ...])
```

If a column in the target table is omitted from the column list, then COPY loads the target column's [DEFAULT \(p. 826\)](#) expression.

If the target column doesn't have a default, then COPY attempts to load NULL.

If COPY attempts to assign NULL to a column that is defined as NOT NULL, the COPY command fails.

If an [IDENTITY \(p. 826\)](#) column is included in the column list, then [EXPLICIT\\_IDS \(p. 709\)](#) must also be specified; if an IDENTITY column is omitted, then EXPLICIT\_IDS can't be specified. If no column list is specified, the command behaves as if a complete, in-order column list was specified, with IDENTITY columns omitted if EXPLICIT\_IDS was also not specified.

If a column is defined with GENERATED BY DEFAULT AS IDENTITY, then it can be copied. Values are generated or updated with values that you supply. The EXPLICIT\_IDS option isn't required. COPY doesn't update the identity high watermark. For more information, see [GENERATED BY DEFAULT AS IDENTITY \(p. 826\)](#).

### JSONPaths file

When loading from data files in JSON or Avro format, COPY automatically maps the data elements in the JSON or Avro source data to the columns in the target table. It does so by matching field names in the Avro schema to column names in the target table or column list.

In some cases, your column names and field names don't match, or you need to map to deeper levels in the data hierarchy. In these cases, you can use a JSONPaths file to explicitly map JSON or Avro data elements to columns.

For more information, see [JSONPaths file \(p. 702\)](#).

### Data format parameters

By default, the COPY command expects the source data to be character-delimited UTF-8 text. The default delimiter is a pipe character (|). If the source data is in another format, use the following parameters to specify the data format:

- [FORMAT \(p. 697\)](#)
- [CSV \(p. 697\)](#)
- [DELIMITER \(p. 697\)](#)
- [FIXEDWIDTH \(p. 697\)](#)
- [SHAPEFILE \(p. 698\)](#)

- [AVRO \(p. 698\)](#)
- [JSON \(p. 700\)](#)
- [PARQUET \(p. 705\)](#)
- [ORC \(p. 705\)](#)

In addition to the standard data formats, COPY supports the following columnar data formats for COPY from Amazon S3:

- [ORC \(p. 705\)](#)
- [PARQUET \(p. 705\)](#)

COPY from columnar format is supported with certain restriction. For more information, see [COPY from columnar data formats \(p. 722\)](#).

## Data format parameters

### FORMAT [AS]

(Optional) Identifies data format keywords. The FORMAT arguments are described following.

#### CSV [ QUOTE [AS] 'quote\_character' ]

Enables use of CSV format in the input data. To automatically escape delimiters, newline characters, and carriage returns, enclose the field in the character specified by the QUOTE parameter. The default quotation mark character is a double quotation mark ( " ). When the quotation mark character is used within a field, escape the character with an additional quotation mark character. For example, if the quotation mark character is a double quotation mark, to insert the string A "quoted" word the input file should include the string "A ""quoted"" word". When the CSV parameter is used, the default delimiter is a comma ( , ). You can specify a different delimiter by using the DELIMITER parameter.

When a field is enclosed in quotation marks, white space between the delimiters and the quotation mark characters is ignored. If the delimiter is a white space character, such as a tab, the delimiter isn't treated as white space.

CSV can't be used with FIXEDWIDTH, REMOVEQUOTES, or ESCAPE.

#### QUOTE [AS] 'quote\_character'

Optional. Specifies the character to be used as the quotation mark character when using the CSV parameter. The default is a double quotation mark ( " ). If you use the QUOTE parameter to define a quotation mark character other than double quotation mark, you don't need to escape double quotation marks within the field. The QUOTE parameter can be used only with the CSV parameter. The AS keyword is optional.

#### DELIMITER [AS] ['delimiter\_char']

Specifies the single ASCII character that is used to separate fields in the input file, such as a pipe character ( | ), a comma ( , ), or a tab ( \t ). Non-printing ASCII characters are supported. ASCII characters can also be represented in octal, using the format '\ddd', where 'd' is an octal digit (0–7). The default delimiter is a pipe character ( | ), unless the CSV parameter is used, in which case the default delimiter is a comma ( , ). The AS keyword is optional. DELIMITER can't be used with FIXEDWIDTH.

#### FIXEDWIDTH 'fixedwidth\_spec'

Loads the data from a file where each column width is a fixed length, rather than columns being separated by a delimiter. The *fixedwidth\_spec* is a string that specifies a user-defined column label and column width. The column label can be either a text string or an integer, depending on what

the user chooses. The column label has no relation to the column name. The order of the label/width pairs must match the order of the table columns exactly. FIXEDWIDTH can't be used with CSV or DELIMITER. In Amazon Redshift, the length of CHAR and VARCHAR columns is expressed in bytes, so be sure that the column width that you specify accommodates the binary length of multibyte characters when preparing the file to be loaded. For more information, see [Character types \(p. 582\)](#).

The format for *fixedwidth\_spec* is shown following:

```
'colLabel1:colWidth1,colLabel2:colWidth2, ...'
```

#### SHAPEFILE [ SIMPLIFY [AUTO] ['tolerance'] ]

Enables use of SHAPEFILE format in the input data. By default, the first column of the shapefile is either a GEOMETRY or IDENTITY column. All subsequent columns follow the order specified in the shapefile.

You can't use SHAPEFILE with FIXEDWIDTH, REMOVEQUOTES, or ESCAPE.

To use GEOGRAPHY objects with COPY FROM SHAPEFILE, first ingest into a GEOMETRY column, and then cast the objects to GEOGRAPHY objects. .

##### SIMPLIFY [*tolerance*]

(Optional) Simplifies all geometries during the ingestion process using the Ramer-Douglas-Peucker algorithm and the given tolerance.

##### SIMPLIFY AUTO [*tolerance*]

(Optional) Simplifies only geometries that are larger than the maximum geometry size. This simplification uses the Ramer-Douglas-Peucker algorithm and the automatically calculated tolerance if this doesn't exceed the specified tolerance. This algorithm calculates the size to store objects within the tolerance specified. The *tolerance* value is optional.

For examples of loading shapefiles, see [Loading a shapefile into Amazon Redshift \(p. 742\)](#).

#### AVRO [AS] '*avro\_option*'

Specifies that the source data is in Avro format.

Avro format is supported for COPY from these services and protocols:

- Amazon S3
- Amazon EMR
- Remote hosts (SSH)

Avro isn't supported for COPY from DynamoDB.

Avro is a data serialization protocol. An Avro source file includes a schema that defines the structure of the data. The Avro schema type must be record. COPY accepts Avro files created using the default uncompressed codec as well as the deflate and snappy compression codecs. For more information about Avro, go to [Apache Avro](#).

Valid values for *avro\_option* are as follows:

- 'auto'
- 'auto ignorecase'
- 's3://*jsonpaths\_file*'

The default is 'auto'.

COPY automatically maps the data elements in the Avro source data to the columns in the target table. It does so by matching field names in the Avro schema to column names in the target table. The matching is case-sensitive for 'auto' and isn't case-sensitive for 'auto ignorecase'.

Column names in Amazon Redshift tables are always lowercase, so when you use the 'auto' option, matching field names must also be lowercase. If the field names aren't all lowercase, you can use the 'auto ignorecase' option. With the default 'auto' argument, COPY recognizes only the first level of fields, or *outer fields*, in the structure.

To explicitly map column names to Avro field names, you can use a [JSONPaths file \(p. 702\)](#).

By default, COPY attempts to match all columns in the target table to Avro field names. To load a subset of the columns, you can optionally specify a column list. If a column in the target table is omitted from the column list, COPY loads the target column's [DEFAULT \(p. 826\)](#) expression. If the target column doesn't have a default, COPY attempts to load NULL. If a column is included in the column list and COPY doesn't find a matching field in the Avro data, COPY attempts to load NULL to the column.

If COPY attempts to assign NULL to a column that is defined as NOT NULL, the COPY command fails.

### Avro Schema

An Avro source data file includes a schema that defines the structure of the data. COPY reads the schema that is part of the Avro source data file to map data elements to target table columns. The following example shows an Avro schema.

```
{
 "name": "person",
 "type": "record",
 "fields": [
 {"name": "id", "type": "int"},
 {"name": "guid", "type": "string"},
 {"name": "name", "type": "string"},
 {"name": "address", "type": "string"}]
}
```

The Avro schema is defined using JSON format. The top-level JSON object contains three name-value pairs with the names, or *keys*, "name", "type", and "fields".

The "fields" key pairs with an array of objects that define the name and data type of each field in the data structure. By default, COPY automatically matches the field names to column names. Column names are always lowercase, so matching field names must also be lowercase, unless you specify the 'auto ignorecase' option. Any field names that don't match a column name are ignored. Order doesn't matter. In the previous example, COPY maps to the column names id, guid, name, and address.

With the default 'auto' argument, COPY matches only the first-level objects to columns. To map to deeper levels in the schema, or if field names and column names don't match, use a JSONPaths file to define the mapping. For more information, see [JSONPaths file \(p. 702\)](#).

If the value associated with a key is a complex Avro data type such as byte, array, record, map, or link, COPY loads the value as a string. Here, the string is the JSON representation of the data. COPY loads Avro enum data types as strings, where the content is the name of the type. For an example, see [COPY from JSON format \(p. 719\)](#).

The maximum size of the Avro file header, which includes the schema and file metadata, is 1 MB.

The maximum size of a single Avro data block is 4 MB. This is distinct from the maximum row size. If the maximum size of a single Avro data block is exceeded, even if the resulting row size is less than the 4 MB row-size limit, the COPY command fails.

In calculating row size, Amazon Redshift internally counts pipe characters ( | ) twice. If your input data contains a very large number of pipe characters, it is possible for row size to exceed 4 MB even if the data block is less than 4 MB.

#### JSON [AS] '*json\_option*'

The source data is in JSON format.

JSON format is supported for COPY from these services and protocols:

- Amazon S3
- COPY from Amazon EMR
- COPY from SSH

JSON isn't supported for COPY from DynamoDB.

Valid values for *json\_option* are as follows :

- 'auto'
- 'auto ignorecase'
- 's3://*jsonpaths\_file*'
- 'noshred'

The default is 'auto'. Amazon Redshift doesn't shred the attributes of JSON structures into multiple columns while loading a JSON document.

By default, COPY attempts to match all columns in the target table to JSON field name keys. To load a subset of the columns, you can optionally specify a column list. If the JSON field name keys aren't all lowercase, you can use the 'auto ignorecase' option or a [JSONPaths file \(p. 702\)](#) to explicitly map column names to JSON field name keys.

If a column in the target table is omitted from the column list, then COPY loads the target column's [DEFAULT \(p. 826\)](#) expression. If the target column doesn't have a default, COPY attempts to load NULL. If a column is included in the column list and COPY doesn't find a matching field in the JSON data, COPY attempts to load NULL to the column.

If COPY attempts to assign NULL to a column that is defined as NOT NULL, the COPY command fails.

COPY maps the data elements in the JSON source data to the columns in the target table. It does so by matching *object keys*, or names, in the source name-value pairs to the names of columns in the target table.

Refer to the following details about each *json\_option* value:

##### 'auto'

With this option, matching is case-sensitive. Column names in Amazon Redshift tables are always lowercase, so when you use the 'auto' option, matching JSON field names must also be lowercase.

##### 'auto ignorecase'

With this option, the matching isn't case-sensitive. Column names in Amazon Redshift tables are always lowercase, so when you use the 'auto ignorecase' option, the corresponding JSON field names can be lowercase, uppercase, or mixed case.

##### 's3://*jsonpaths\_file*'

With this option, COPY uses the named JSONPaths file to map the data elements in the JSON source data to the columns in the target table. The *s3://jsonpaths\_file* argument must be an Amazon S3 object key that explicitly references a single file. An example is 's3://mybucket/jsonpaths.txt'. The argument can't be a key prefix. For more information about using a JSONPaths file, see [the section called "JSONPaths file" \(p. 702\)](#).

In some cases, the file specified by `jsonpaths_file` has the same prefix as the path specified by `copy_from_s3_objectpath` for the data files. If so, COPY reads the JSONPaths file as a data file and returns errors. For example, suppose that your data files use the object path `s3://mybucket/my_data.json` and your JSONPaths file is `s3://mybucket/my_data.jsonpaths`. In this case, COPY attempts to load `my_data.jsonpaths` as a data file. `'noshred'`

With this option, Amazon Redshift doesn't shred the attributes of JSON structures into multiple columns while loading a JSON document.

### JSON data file

The JSON data file contains a set of either objects or arrays. COPY loads each JSON object or array into one row in the target table. Each object or array corresponding to a row must be a stand-alone, root-level structure; that is, it must not be a member of another JSON structure.

A JSON *object* begins and ends with braces (`{ }`) and contains an unordered collection of name-value pairs. Each paired name and value are separated by a colon, and the pairs are separated by commas. By default, the *object key*, or name, in the name-value pairs must match the name of the corresponding column in the table. Column names in Amazon Redshift tables are always lowercase, so matching JSON field name keys must also be lowercase. If your column names and JSON keys don't match, use a [the section called "JSONPaths file" \(p. 702\)](#) to explicitly map columns to keys.

Order in a JSON object doesn't matter. Any names that don't match a column name are ignored. The following shows the structure of a simple JSON object.

```
{
 "column1": "value1",
 "column2": value2,
 "notacolumn" : "ignore this value"
}
```

A JSON *array* begins and ends with brackets (`[ ]`), and contains an ordered collection of values separated by commas. If your data files use arrays, you must specify a JSONPaths file to match the values to columns. The following shows the structure of a simple JSON array.

```
["value1", value2]
```

The JSON must be well-formed. For example, the objects or arrays can't be separated by commas or any other characters except white space. Strings must be enclosed in double quotation mark characters. Quote characters must be simple quotation marks (0x22), not slanted or "smart" quotation marks.

The maximum size of a single JSON object or array, including braces or brackets, is 4 MB. This is distinct from the maximum row size. If the maximum size of a single JSON object or array is exceeded, even if the resulting row size is less than the 4 MB row-size limit, the COPY command fails.

In calculating row size, Amazon Redshift internally counts pipe characters (`|`) twice. If your input data contains a very large number of pipe characters, it is possible for row size to exceed 4 MB even if the object size is less than 4 MB.

COPY loads `\n` as a newline character and loads `\t` as a tab character. To load a backslash, escape it with a backslash (`\\"`).

COPY searches the specified JSON source for a well-formed, valid JSON object or array. If COPY encounters any non-white-space characters before locating a usable JSON structure, or between valid JSON objects or arrays, COPY returns an error for each instance. These errors count toward the MAXERROR error count. When the error count equals or exceeds MAXERROR, COPY fails.

For each error, Amazon Redshift records a row in the `STL_LOAD_ERRORS` system table. The `LINE_NUMBER` column records the last line of the JSON object that caused the error.

If `IGNOREHEADER` is specified, `COPY` ignores the specified number of lines in the JSON data. Newline characters in the JSON data are always counted for `IGNOREHEADER` calculations.

`COPY` loads empty strings as empty fields by default. If `EMPTYASNULL` is specified, `COPY` loads empty strings for `CHAR` and `VARCHAR` fields as `NULL`. Empty strings for other data types, such as `INT`, are always loaded with `NULL`.

The following options aren't supported with JSON:

- `CSV`
- `DELIMITER`
- `ESCAPE`
- `FILLRECORD`
- `FIXEDWIDTH`
- `IGNOREBLANKLINES`
- `NULL AS`
- `READRATIO`
- `REMOVEQUOTES`

For more information, see [COPY from JSON format \(p. 719\)](#). For more information about JSON data structures, go to [www.json.org](http://www.json.org).

### JSONPaths file

If you are loading from JSON-formatted or Avro source data, by default `COPY` maps the first-level data elements in the source data to the columns in the target table. It does so by matching each name, or object key, in a name-value pair to the name of a column in the target table.

If your column names and object keys don't match, or to map to deeper levels in the data hierarchy, you can use a JSONPaths file to explicitly map JSON or Avro data elements to columns. The JSONPaths file maps JSON data elements to columns by matching the column order in the target table or column list.

The JSONPaths file must contain only a single JSON object (not an array). The JSON object is a name-value pair. The *object key*, which is the name in the name-value pair, must be "jsonpaths". The *value* in the name-value pair is an array of *JSONPath expressions*. Each JSONPath expression references a single element in the JSON data hierarchy or Avro schema, similarly to how an XPath expression refers to elements in an XML document. For more information, see [JSONPath expressions \(p. 703\)](#).

To use a JSONPaths file, add the `JSON` or `AVRO` keyword to the `COPY` command. Specify the S3 bucket name and object path of the JSONPaths file using the following format.

```
COPY tablename
FROM 'data_source'
CREDENTIALS 'credentials-args'
FORMAT AS { AVRO | JSON } 's3://jsonpaths_file';
```

The `s3://jsonpaths_file` value must be an Amazon S3 object key that explicitly references a single file, such as `'s3://mybucket/jsonpaths.txt'`. It can't be a key prefix.

In some cases, if you're loading from Amazon S3 the file specified by `jsonpaths_file` has the same prefix as the path specified by `copy_from_s3_objectpath` for the data files. If so, `COPY` reads the JSONPaths file as a data file and returns errors. For example, suppose that your data files use

the object path `s3://mybucket/my_data.json` and your JSONPaths file is `s3://mybucket/my_data.jsonpaths`. In this case, COPY attempts to load `my_data.jsonpaths` as a data file.

If the key name is any string other than "jsonpaths", the COPY command doesn't return an error, but it ignores `jsonpaths_file` and uses the 'auto' argument instead.

If any of the following occurs, the COPY command fails:

- The JSON is malformed.
- There is more than one JSON object.
- Any characters except white space exist outside the object.
- An array element is an empty string or isn't a string.

MAXERROR doesn't apply to the JSONPaths file.

The JSONPaths file must not be encrypted, even if the [ENCRYPTED \(p. 686\)](#) option is specified.

For more information, see [COPY from JSON format \(p. 719\)](#).

### JSONPath expressions

The JSONPaths file uses JSONPath expressions to map data fields to target columns. Each JSONPath expression corresponds to one column in the Amazon Redshift target table. The order of the JSONPath array elements must match the order of the columns in the target table or the column list, if a column list is used.

The double quotation mark characters are required as shown, both for the field names and the values. The quotation mark characters must be simple quotation marks (0x22), not slanted or "smart" quotation marks.

If an object element referenced by a JSONPath expression isn't found in the JSON data, COPY attempts to load a NULL value. If the referenced object is malformed, COPY returns a load error.

If an array element referenced by a JSONPath expression isn't found in the JSON or Avro data, COPY fails with the following error: Invalid JSONPath format: Not an array or index out of range. Remove any array elements from the JSONPaths that don't exist in the source data and verify that the arrays in the source data are well formed.

The JSONPath expressions can use either bracket notation or dot notation, but you can't mix notations. The following example shows JSONPath expressions using bracket notation.

```
{
 "jsonpaths": [
 "$['venuename']",
 "$['venuecity']",
 "$['venuestate']",
 "$['venueseats']"
]
}
```

The following example shows JSONPath expressions using dot notation.

```
{
 "jsonpaths": [
 ".venuename",
 ".venuecity",
 ".venuestate",
 ".venueseats"
]
}
```

```
]
}
```

In the context of Amazon Redshift COPY syntax, a JSONPath expression must specify the explicit path to a single name element in a JSON or Avro hierarchical data structure. Amazon Redshift doesn't support any JSONPath elements, such as wildcard characters or filter expressions, that might resolve to an ambiguous path or multiple name elements.

For more information, see [COPY from JSON format \(p. 719\)](#).

### Using JSONPaths with Avro Data

The following example shows an Avro schema with multiple levels.

```
{
 "name": "person",
 "type": "record",
 "fields": [
 {"name": "id", "type": "int"},
 {"name": "guid", "type": "string"},
 {"name": "isActive", "type": "boolean"},
 {"name": "age", "type": "int"},
 {"name": "name", "type": "string"},
 {"name": "address", "type": "string"},
 {"name": "latitude", "type": "double"},
 {"name": "longitude", "type": "double"},
 {
 "name": "tags",
 "type": {
 "type" : "array",
 "name" : "inner_tags",
 "items" : "string"
 }
 },
 {
 "name": "friends",
 "type": {
 "type" : "array",
 "name" : "inner_friends",
 "items" : {
 "name" : "friends_record",
 "type" : "record",
 "fields": [
 {"name" : "id", "type" : "int"},
 {"name" : "name", "type" : "string"}
]
 }
 }
 },
 {"name": "randomArrayItem", "type": "string"}
]
}
```

The following example shows a JSONPaths file that uses AvroPath expressions to reference the previous schema.

```
{
 "jsonpaths": [
 "$.id",
 "$.guid",
 "$.address",
 "$.friends[0].id"
```

```
]
}
```

The JSONPaths example includes the following elements:

#### jsonpaths

The name of the JSON object that contains the AvroPath expressions.

```
[...]
```

Brackets enclose the JSON array that contains the path elements.

```
$
```

The dollar sign refers to the root element in the Avro schema, which is the "fields" array.

```
("$.id",
```

The target of the AvroPath expression. In this instance, the target is the element in the "fields" array with the name "id". The expressions are separated by commas.

```
"$.friends[0].id"
```

Brackets indicate an array index. JSONPath expressions use zero-based indexing, so this expression references the first element in the "friends" array with the name "id".

The Avro schema syntax requires using *inner fields* to define the structure of record and array data types. The inner fields are ignored by the AvroPath expressions. For example, the field "friends" defines an array named "inner\_friends", which in turn defines a record named "friends\_record". The AvroPath expression to reference the field "id" can ignore the extra fields to reference the target field directly. The following AvroPath expressions reference the two fields that belong to the "friends" array.

```
"$.friends[0].id"
"$.friends[0].name"
```

#### Columnar data format parameters

In addition to the standard data formats, COPY supports the following columnar data formats for COPY from Amazon S3. COPY from columnar format is supported with certain restrictions. For more information, see [COPY from columnar data formats \(p. 722\)](#).

##### ORC

Loads the data from a file that uses Optimized Row Columnar (ORC) file format.

##### PARQUET

Loads the data from a file that uses Parquet file format.

#### File compression parameters

You can load from compressed data files by specifying the following parameters.

#### File compression parameters

##### BZIP2

A value that specifies that the input file or files are in compressed bzip2 format (.bz2 files). The COPY operation reads each compressed file and uncompresses the data as it loads.

#### GZIP

A value that specifies that the input file or files are in compressed gzip format (.gz files). The COPY operation reads each compressed file and uncompresses the data as it loads.

#### LZOP

A value that specifies that the input file or files are in compressed lzop format (.lzo files). The COPY operation reads each compressed file and uncompresses the data as it loads.

**Note**

COPY doesn't support files that are compressed using the lzop --filter option.

#### ZSTD

A value that specifies that the input file or files are in compressed Zstandard format (.zst files). The COPY operation reads each compressed file and uncompresses the data as it loads. For more information, see [ZSTD \(p. 45\)](#).

**Note**

ZSTD is supported only with COPY from Amazon S3.

### Data conversion parameters

As it loads the table, COPY attempts to implicitly convert the strings in the source data to the data type of the target column. If you need to specify a conversion that is different from the default behavior, or if the default conversion results in errors, you can manage data conversions by specifying the following parameters. For more information on the syntax of these parameters, see [COPY syntax](#).

- [ACCEPTANYDATE \(p. 706\)](#)
- [ACCEPTINVCHARS \(p. 707\)](#)
- [BLANKSASNLL \(p. 707\)](#)
- [DATEFORMAT \(p. 707\)](#)
- [EMPTYASNLL \(p. 707\)](#)
- [ENCODING \(p. 707\)](#)
- [ESCAPE \(p. 708\)](#)
- [EXPLICIT\\_IDS \(p. 709\)](#)
- [FILLRECORD \(p. 709\)](#)
- [IGNOREBLANKLINES \(p. 709\)](#)
- [IGNOREHEADER \(p. 710\)](#)
- [NULL AS \(p. 710\)](#)
- [REMOVEQUOTES \(p. 710\)](#)
- [ROUNDDEC \(p. 710\)](#)
- [TIMEFORMAT \(p. 710\)](#)
- [TRIMBLANKS \(p. 711\)](#)
- [TRUNCATECOLUMNS \(p. 711\)](#)

### Data conversion parameters

#### ACCEPTANYDATE

Allows any date format, including invalid formats such as `00/00/00 00:00:00`, to be loaded without generating an error. This parameter applies only to `TIMESTAMP` and `DATE` columns. Always use `ACCEPTANYDATE` with the `DATEFORMAT` parameter. If the date format for the data doesn't match the `DATEFORMAT` specification, Amazon Redshift inserts a `NULL` value into that field.

#### ACCEPTINVCHARS [AS] ['replacement\_char']

Enables loading of data into VARCHAR columns even if the data contains invalid UTF-8 characters. When ACCEPTINVCHARS is specified, COPY replaces each invalid UTF-8 character with a string of equal length consisting of the character specified by *replacement\_char*. For example, if the replacement character is '^', an invalid three-byte character will be replaced with '^^^'.

The replacement character can be any ASCII character except NULL. The default is a question mark (?). For information about invalid UTF-8 characters, see [Multibyte character load errors \(p. 92\)](#).

COPY returns the number of rows that contained invalid UTF-8 characters, and it adds an entry to the [STL\\_REPLACEMENTS \(p. 1547\)](#) system table for each affected row, up to a maximum of 100 rows for each node slice. Additional invalid UTF-8 characters are also replaced, but those replacement events aren't recorded.

If ACCEPTINVCHARS isn't specified, COPY returns an error whenever it encounters an invalid UTF-8 character.

ACCEPTINVCHARS is valid only for VARCHAR columns.

#### BLANKSASNULL

Loads blank fields, which consist of only white space characters, as NULL. This option applies only to CHAR and VARCHAR columns. Blank fields for other data types, such as INT, are always loaded with NULL. For example, a string that contains three space characters in succession (and no other characters) is loaded as a NULL. The default behavior, without this option, is to load the space characters as is.

#### DATEFORMAT [AS] {'dateformat\_string' | 'auto' }

If no DATEFORMAT is specified, the default format is 'YYYY-MM-DD'. For example, an alternative valid format is 'MM-DD-YYYY'.

If the COPY command doesn't recognize the format of your date or time values, or if your date or time values use different formats, use the 'auto' argument with the DATEFORMAT or TIMEFORMAT parameter. The 'auto' argument recognizes several formats that aren't supported when using a DATEFORMAT and TIMEFORMAT string. The 'auto' keyword is case-sensitive. For more information, see [Using automatic recognition with DATEFORMAT and TIMEFORMAT \(p. 725\)](#).

The date format can include time information (hour, minutes, seconds), but this information is ignored. The AS keyword is optional. For more information, see [DATEFORMAT and TIMEFORMAT strings \(p. 724\)](#).

#### EMPTYASNULL

Indicates that Amazon Redshift should load empty CHAR and VARCHAR fields as NULL. Empty fields for other data types, such as INT, are always loaded with NULL. Empty fields occur when data contains two delimiters in succession with no characters between the delimiters. EMPTYASNULL and NULL AS '' (empty string) produce the same behavior.

#### ENCODING [AS] *file\_encoding*

Specifies the encoding type of the load data. The COPY command converts the data from the specified encoding into UTF-8 during loading.

Valid values for *file\_encoding* are as follows:

- UTF8
- UTF16
- UTF16LE
- UTF16BE

The default is UTF8.

Source file names must use UTF-8 encoding.

The following files must use UTF-8 encoding, even if a different encoding is specified for the load data:

- Manifest files
- JSONPaths files

The argument strings provided with the following parameters must use UTF-8:

- FIXEDWIDTH '*fixedwidth\_spec*'
- ACCEPTINVCHARS '*replacement\_char*'
- DATEFORMAT '*dateformat\_string*'
- TIMEFORMAT '*timeformat\_string*'
- NULL AS '*null\_string*'

Fixed-width data files must use UTF-8 encoding. The field widths are based on the number of characters, not the number of bytes.

All load data must use the specified encoding. If COPY encounters a different encoding, it skips the file and returns an error.

If you specify UTF16, then your data must have a byte order mark (BOM). If you know whether your UTF-16 data is little-endian (LE) or big-endian (BE), you can use UTF16LE or UTF16BE, regardless of the presence of a BOM.

#### ESCAPE

When this parameter is specified, the backslash character (\) in input data is treated as an escape character. The character that immediately follows the backslash character is loaded into the table as part of the current column value, even if it is a character that normally serves a special purpose. For example, you can use this parameter to escape the delimiter character, a quotation mark, an embedded newline character, or the escape character itself when any of these characters is a legitimate part of a column value.

If you specify the ESCAPE parameter in combination with the REMOVEQUOTES parameter, you can escape and retain quotation marks (' or ") that might otherwise be removed. The default null string, \N, works as is, but it can also be escaped in the input data as \\N. As long as you don't specify an alternative null string with the NULL AS parameter, \N and \\N produce the same results.

#### Note

The control character 0x00 (NUL) can't be escaped and should be removed from the input data or converted. This character is treated as an end of record (EOR) marker, causing the remainder of the record to be truncated.

You can't use the ESCAPE parameter for FIXEDWIDTH loads, and you can't specify the escape character itself; the escape character is always the backslash character. Also, you must ensure that the input data contains the escape character in the appropriate places.

Here are some examples of input data and the resulting loaded data when the ESCAPE parameter is specified. The result for row 4 assumes that the REMOVEQUOTES parameter is also specified. The input data consists of two pipe-delimited fields:

```
1|The quick brown fox\[newline]
jumped over the lazy dog.
2| A\\B\\C
3| A \| B \| C
```

```
4| 'A Midsummer Night\'s Dream'
```

The data loaded into column 2 looks like this:

```
The quick brown fox
jumped over the lazy dog.
A\B\C
A|B|C
A Midsummer Night's Dream
```

**Note**

Applying the escape character to the input data for a load is the responsibility of the user. One exception to this requirement is when you reload data that was previously unloaded with the ESCAPE parameter. In this case, the data will already contain the necessary escape characters.

The ESCAPE parameter doesn't interpret octal, hex, Unicode, or other escape sequence notation. For example, if your source data contains the octal line feed value (\012) and you try to load this data with the ESCAPE parameter, Amazon Redshift loads the value 012 into the table and doesn't interpret this value as a line feed that is being escaped.

In order to escape newline characters in data that originates from Microsoft Windows platforms, you might need to use two escape characters: one for the carriage return and one for the line feed. Alternatively, you can remove the carriage returns before loading the file (for example, by using the dos2unix utility).

#### EXPLICIT\_IDS

Use EXPLICIT\_IDS with tables that have IDENTITY columns if you want to override the autogenerated values with explicit values from the source data files for the tables. If the command includes a column list, that list must include the IDENTITY columns to use this parameter. The data format for EXPLICIT\_IDS values must match the IDENTITY format specified by the CREATE TABLE definition.

When you run a COPY command against a table with the EXPLICIT\_IDS option, Amazon Redshift does not check the uniqueness of IDENTITY columns in the table.

If a column is defined with GENERATED BY DEFAULT AS IDENTITY, then it can be copied. Values are generated or updated with values that you supply. The EXPLICIT\_IDS option isn't required. COPY doesn't update the identity high watermark.

For an example of a COPY command using EXPLICIT\_IDS, see [Load VENUE with explicit values for an IDENTITY column \(p. 731\)](#).

#### FILLRECORD

Allows data files to be loaded when contiguous columns are missing at the end of some of the records. The missing columns are loaded as NULLs. For text and CSV formats, if the missing column is a VARCHAR column, zero-length strings are loaded instead of NULLs. To load NULLs to VARCHAR columns from text and CSV, specify the EMPTYASNULL keyword. NULL substitution only works if the column definition allows NULLs.

For example, if the table definition contains four nullable CHAR columns, and a record contains the values apple, orange, banana, mango, the COPY command could load and fill in a record that contains only the values apple, orange. The missing CHAR values would be loaded as NULL values.

#### IGNOREBLANKLINES

Ignores blank lines that only contain a line feed in a data file and does not try to load them.

#### **IGNOREHEADER [ AS ] *number\_rows***

Treats the specified *number\_rows* as a file header and doesn't load them. Use IGNOREHEADER to skip file headers in all files in a parallel load.

#### **NULL AS '*null\_string*'**

Loads fields that match *null\_string* as NULL, where *null\_string* can be any string. If your data includes a null terminator, also referred to as NUL (UTF-8 0000) or binary zero (0x000), COPY treats it as any other character. For example, a record containing '1' || NUL || '2' is copied as string of length 3 bytes. If a field contains only NUL, you can use NULL AS to replace the null terminator with NULL by specifying '\0' or '\000'—for example, NULL AS '\0' or NULL AS '\000'. If a field contains a string that ends with NUL and NULL AS is specified, the string is inserted with NUL at the end. Do not use '\n' (newline) for the *null\_string* value. Amazon Redshift reserves '\n' for use as a line delimiter. The default *null\_string* is '\N'.

#### **Note**

If you attempt to load nulls into a column defined as NOT NULL, the COPY command will fail.

#### **REMOVEQUOTES**

Removes surrounding quotation marks from strings in the incoming data. All characters within the quotation marks, including delimiters, are retained. If a string has a beginning single or double quotation mark but no corresponding ending mark, the COPY command fails to load that row and returns an error. The following table shows some simple examples of strings that contain quotation marks and the resulting loaded values.

| <b>Input String</b>                     | <b>Loaded Value with REMOVEQUOTES Option</b> |
|-----------------------------------------|----------------------------------------------|
| "The delimiter is a pipe ( ) character" | The delimiter is a pipe ( ) character        |
| 'Black'                                 | Black                                        |
| "White"                                 | White                                        |
| Blue'                                   | Blue'                                        |
| 'Blue                                   | <i>Value not loaded: error condition</i>     |
| "Blue                                   | <i>Value not loaded: error condition</i>     |
| '''Black'''                             | ' Black '                                    |
| ''                                      | <white space>                                |

#### **ROUNDDEC**

Rounds up numeric values when the scale of the input value is greater than the scale of the column. By default, COPY truncates values when necessary to fit the scale of the column. For example, if a value of 20.259 is loaded into a DECIMAL(8,2) column, COPY truncates the value to 20.25 by default. If ROUNDDEC is specified, COPY rounds the value to 20.26. The INSERT command always rounds values when necessary to match the column's scale, so a COPY command with the ROUNDDEC parameter behaves the same as an INSERT command.

#### **TIMEFORMAT [AS] {'timeformat\_string' | 'auto' | 'epochsecs' | 'epochmillisecs' }**

Specifies the time format. If no TIMEFORMAT is specified, the default format is YYYY-MM-DD HH:MI:SS for TIMESTAMP columns or YYYY-MM-DD HH:MI:SSOF for Timestamptz columns, where OF is the offset from Coordinated Universal Time (UTC). You can't include a time zone specifier in the *timeformat\_string*. To load Timestamptz data that is in a format different from

the default format, specify 'auto'; for more information, see [Using automatic recognition with DATEFORMAT and TIMEFORMAT \(p. 725\)](#). For more information about *timeformat\_string*, see [DATEFORMAT and TIMEFORMAT strings \(p. 724\)](#).

The 'auto' argument recognizes several formats that aren't supported when using a DATEFORMAT and TIMEFORMAT string. If the COPY command doesn't recognize the format of your date or time values, or if your date and time values use formats different from each other, use the 'auto' argument with the DATEFORMAT or TIMEFORMAT parameter. For more information, see [Using automatic recognition with DATEFORMAT and TIMEFORMAT \(p. 725\)](#).

If your source data is represented as epoch time, that is the number of seconds or milliseconds since January 1, 1970, 00:00:00 UTC, specify 'epochsecs' or 'epochmillisecs'.

The 'auto', 'epochsecs', and 'epochmillisecs' keywords are case-sensitive.

The AS keyword is optional.

#### TRIMBLANKS

Removes the trailing white space characters from a VARCHAR string. This parameter applies only to columns with a VARCHAR data type.

#### TRUNCATECOLUMNS

Truncates data in columns to the appropriate number of characters so that it fits the column specification. Applies only to columns with a VARCHAR or CHAR data type, and rows 4 MB or less in size.

## Data load operations

Manage the default behavior of the load operation for troubleshooting or to reduce load times by specifying the following parameters.

- [COMPROWS \(p. 711\)](#)
- [COMPUPDATE \(p. 712\)](#)
- [IGNOREALLERRORS \(p. 712\)](#)
- [MAXERROR \(p. 712\)](#)
- [NOLOAD \(p. 712\)](#)
- [STATUPDATE \(p. 713\)](#)

### Parameters

#### COMPROWS *numrows*

Specifies the number of rows to be used as the sample size for compression analysis. The analysis is run on rows from each data slice. For example, if you specify COMPROWS 1000000 (1,000,000) and the system contains four total slices, no more than 250,000 rows for each slice are read and analyzed.

If COMPROWS isn't specified, the sample size defaults to 100,000 for each slice. Values of COMPROWS lower than the default of 100,000 rows for each slice are automatically upgraded to the default value. However, automatic compression will not take place if the amount of data being loaded is insufficient to produce a meaningful sample.

If the COMPROWS number is greater than the number of rows in the input file, the COPY command still proceeds and runs the compression analysis on all of the available rows. The accepted range for this argument is a number between 1000 and 2147483647 (2,147,483,647).

**COMPUPDATE [ PRESET | { ON | TRUE } | { OFF | FALSE } ],**

Controls whether compression encodings are automatically applied during a COPY.

When COMPUPDATE is PRESET, the COPY command chooses the compression encoding for each column if the target table is empty; even if the columns already have encodings other than RAW. Currently specified column encodings can be replaced. Encoding for each column is based on the column data type. No data is sampled. Amazon Redshift automatically assigns compression encoding as follows:

- Columns that are defined as sort keys are assigned RAW compression.
- Columns that are defined as BOOLEAN, REAL, or DOUBLE PRECISION data types are assigned RAW compression.
- Columns that are defined as SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIMESTAMP, or TIMESTAMPTZ are assigned AZ64 compression.
- Columns that are defined as CHAR or VARCHAR are assigned LZO compression.

When COMPUPDATE is omitted, the COPY command chooses the compression encoding for each column only if the target table is empty and you have not specified an encoding (other than RAW) for any of the columns. The encoding for each column is determined by Amazon Redshift. No data is sampled.

When COMPUPDATE is ON (or TRUE), or COMPUPDATE is specified without an option, the COPY command applies automatic compression if the table is empty; even if the table columns already have encodings other than RAW. Currently specified column encodings can be replaced. Encoding for each column is based on an analysis of sample data. For more information, see [Loading tables with automatic compression \(p. 87\)](#).

When COMPUPDATE is OFF (or FALSE), automatic compression is disabled. Column encodings aren't changed.

For information about the system table to analyze compression, see [STL\\_ANALYZE\\_COMPRESSION \(p. 1501\)](#).

#### IGNOREALLERRORS

You can specify this option to ignore all errors that occur during the load operation.

You can't specify the IGNOREALLERRORS option if you specify the MAXERROR option. You can't specify the IGNOREALLERRORS option for columnar formats including ORC and Parquet.

#### MAXERROR [AS] *error\_count*

If the load returns the *error\_count* number of errors or greater, the load fails. If the load returns fewer errors, it continues and returns an INFO message that states the number of rows that could not be loaded. Use this parameter to allow loads to continue when certain rows fail to load into the table because of formatting errors or other inconsistencies in the data.

Set this value to 0 or 1 if you want the load to fail as soon as the first error occurs. The AS keyword is optional. The MAXERROR default value is 0 and the limit is 100000.

The actual number of errors reported might be greater than the specified MAXERROR because of the parallel nature of Amazon Redshift. If any node in the Amazon Redshift cluster detects that MAXERROR has been exceeded, each node reports all of the errors it has encountered.

#### NOLOAD

Checks the validity of the data file without actually loading the data. Use the NOLOAD parameter to make sure that your data file loads without any errors before running the actual data load. Running COPY with the NOLOAD parameter is much faster than loading the data because it only parses the files.

## STATUPDATE [ { ON | TRUE } | { OFF | FALSE } ]

Governs automatic computation and refresh of optimizer statistics at the end of a successful COPY command. By default, if the STATUPDATE parameter isn't used, statistics are updated automatically if the table is initially empty.

Whenever ingesting data into a nonempty table significantly changes the size of the table, we recommend updating statistics either by running an [ANALYZE \(p. 663\)](#) command or by using the STATUPDATE ON argument.

With STATUPDATE ON (or TRUE), statistics are updated automatically regardless of whether the table is initially empty. If STATUPDATE is used, the current user must be either the table owner or a superuser. If STATUPDATE is not specified, only INSERT permission is required.

With STATUPDATE OFF (or FALSE), statistics are never updated.

For additional information, see [Analyzing tables \(p. 105\)](#).

## Alphabetical parameter list

The following list provides links to each COPY command parameter description, sorted alphabetically.

- [ACCEPTANYDATE \(p. 706\)](#)
- [ACCEPTINVCHARS \(p. 707\)](#)
- [ACCESS\\_KEY\\_ID and SECRET\\_ACCESS\\_KEY \(p. 694\)](#)
- [AVRO \(p. 698\)](#)
- [BLANKSASNULL \(p. 707\)](#)
- [BZIP2 \(p. 705\)](#)
- [COMPROWS \(p. 711\)](#)
- [COMPUPDATE \(p. 712\)](#)
- [CREDENTIALS \(p. 694\)](#)
- [CSV \(p. 697\)](#)
- [DATEFORMAT \(p. 707\)](#)
- [DELIMITER \(p. 697\)](#)
- [EMPTYASNULL \(p. 707\)](#)
- [ENCODING \(p. 707\)](#)
- [ENCRYPTED \(p. 686\)](#)
- [ESCAPE \(p. 708\)](#)
- [EXPLICIT\\_IDS \(p. 709\)](#)
- [FILLRECORD \(p. 709\)](#)
- [FIXEDWIDTH \(p. 697\)](#)
- [FORMAT \(p. 697\)](#)
- [FROM \(p. 684\)](#)
- [GZIP \(p. 706\)](#)
- [IAM\\_ROLE \(p. 694\)](#)
- [IGNOREALLERRORS \(p. 712\)](#)
- [IGNOREBLANKLINES \(p. 709\)](#)
- [IGNOREHEADER \(p. 710\)](#)
- [JSON \(p. 700\)](#)
- [LZOP \(p. 706\)](#)

- [MANIFEST \(p. 686\)](#)
- [MASTER\\_SYMMETRIC\\_KEY \(p. 686\)](#)
- [MAXERROR \(p. 712\)](#)
- [NOLOAD \(p. 712\)](#)
- [NULL AS \(p. 710\)](#)
- [READRATIO \(p. 692\)](#)
- [REGION \(p. 686\)](#)
- [REMOVEQUOTES \(p. 710\)](#)
- [ROUNDDEC \(p. 710\)](#)
- [SESSION\\_TOKEN \(p. 694\)](#)
- [SHAPEFILE \(p. 698\)](#)
- [SSH \(p. 691\)](#)
- [STATUPDATE \(p. 713\)](#)
- [TIMEFORMAT \(p. 710\)](#)
- [SESSION\\_TOKEN \(p. 694\)](#)
- [TRIMBLANKS \(p. 711\)](#)
- [TRUNCATECOLUMNS \(p. 711\)](#)
- [ZSTD \(p. 706\)](#)

## Usage notes

### Topics

- [Permissions to access other AWS Resources \(p. 714\)](#)
- [Using COPY with Amazon S3 access point aliases \(p. 718\)](#)
- [Loading multibyte data from Amazon S3 \(p. 718\)](#)
- [Loading a column of the GEOMETRY or GEOGRAPHY data type \(p. 718\)](#)
- [Loading the HLLSKETCH data type \(p. 718\)](#)
- [Loading a column of the VARBYTE data type \(p. 719\)](#)
- [Errors when reading multiple files \(p. 719\)](#)
- [COPY from JSON format \(p. 719\)](#)
- [COPY from columnar data formats \(p. 722\)](#)
- [DATEFORMAT and TIMEFORMAT strings \(p. 724\)](#)
- [Using automatic recognition with DATEFORMAT and TIMEFORMAT \(p. 725\)](#)

### Permissions to access other AWS Resources

To move data between your cluster and another AWS resource, such as Amazon S3, Amazon DynamoDB, Amazon EMR, or Amazon EC2, your cluster must have permission to access the resource and perform the necessary actions. For example, to load data from Amazon S3, COPY must have LIST access to the bucket and GET access for the bucket objects. For information about minimum permissions, see [IAM permissions for COPY, UNLOAD, and CREATE LIBRARY \(p. 718\)](#).

To get authorization to access the resource, your cluster must be authenticated. You can choose either of the following authentication methods:

- [Role-based access control \(p. 715\)](#) – For role-based access control, you specify an AWS Identity and Access Management (IAM) role that your cluster uses for authentication and authorization.

To safeguard your AWS credentials and sensitive data, we strongly recommend using role-based authentication.

- [Key-based access control \(p. 716\)](#) – For key-based access control, you provide the AWS access credentials (access key ID and secret access key) for a user as plain text.

### Role-based access control

With role-based access control, your cluster temporarily assumes an IAM role on your behalf. Then, based on the authorizations granted to the role, your cluster can access the required AWS resources.

Creating an IAM *role* is similar to granting permissions to a user, in that it is an AWS identity with permissions policies that determine what the identity can and can't do in AWS. However, instead of being uniquely associated with one user, a role can be assumed by any entity that needs it. Also, a role doesn't have any credentials (a password or access keys) associated with it. Instead, if a role is associated with a cluster, access keys are created dynamically and provided to the cluster.

We recommend using role-based access control because it provides more secure, fine-grained control of access to AWS resources and sensitive user data, in addition to safeguarding your AWS credentials.

Role-based authentication delivers the following benefits:

- You can use AWS standard IAM tools to define an IAM role and associate the role with multiple clusters. When you modify the access policy for a role, the changes are applied automatically to all clusters that use the role.
- You can define fine-grained IAM policies that grant permissions for specific clusters and database users to access specific AWS resources and actions.
- Your cluster obtains temporary session credentials at run time and refreshes the credentials as needed until the operation completes. If you use key-based temporary credentials, the operation fails if the temporary credentials expire before it completes.
- Your access key ID and secret access key ID aren't stored or transmitted in your SQL code.

To use role-based access control, you must first create an IAM role using the Amazon Redshift service role type, and then attach the role to your cluster. The role must have, at a minimum, the permissions listed in [IAM permissions for COPY, UNLOAD, and CREATE LIBRARY \(p. 718\)](#). For steps to create an IAM role and attach it to your cluster, see [Authorizing Amazon Redshift to Access Other AWS Services On Your Behalf](#) in the *Amazon Redshift Management Guide*.

You can add a role to a cluster or view the roles associated with a cluster by using the Amazon Redshift Management Console, CLI, or API. For more information, see [Associating an IAM Role With a Cluster](#) in the *Amazon Redshift Management Guide*.

When you create an IAM role, IAM returns an Amazon Resource Name (ARN) for the role. To specify an IAM role, provide the role ARN with either the [IAM\\_ROLE \(p. 694\)](#) parameter or the [CREDENTIALS \(p. 694\)](#) parameter.

For example, suppose the following role is attached to the cluster.

```
"IamRoleArn": "arn:aws:iam::0123456789012:role/MyRedshiftRole"
```

The following COPY command example uses the IAM\_ROLE parameter with the ARN in the previous example for authentication and access to Amazon S3.

```
copy customer from 's3://mybucket/mydata'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

The following COPY command example uses the CREDENTIALS parameter to specify the IAM role.

```
copy customer from 's3://mybucket/mydata'
credentials
'aws_iam_role=arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

In addition, a superuser can grant the ASSUMEROLE privilege to database users and groups to provide access to a role for COPY operations. For information, see [GRANT \(p. 887\)](#).

#### Key-based access control

With key-based access control, you provide the access key ID and secret access key for an IAM user that is authorized to access the AWS resources that contain the data. You can user either the [ACCESS\\_KEY\\_ID](#) and [SECRET\\_ACCESS\\_KEY \(p. 694\)](#) parameters together or the [CREDENTIALS \(p. 694\)](#) parameter.

##### Note

We strongly recommend using an IAM role for authentication instead of supplying a plain-text access key ID and secret access key. If you choose key-based access control, never use your AWS account (root) credentials. Always create an IAM user and provide that user's access key ID and secret access key. For steps to create an IAM user, see [Creating an IAM User in Your AWS Account](#).

To authenticate using ACCESS\_KEY\_ID and SECRET\_ACCESS\_KEY, replace `<access-key-id>` and `<secret-access-key>` with an authorized user's access key ID and full secret access key as shown following.

```
ACCESS_KEY_ID '<access-key-id>'
SECRET_ACCESS_KEY '<secret-access-key>;'
```

To authenticate using the CREDENTIALS parameter, replace `<access-key-id>` and `<secret-access-key>` with an authorized user's access key ID and full secret access key as shown following.

```
CREDENTIALS
'aws_access_key_id=<access-key-id>;aws_secret_access_key=<secret-access-key>;'
```

The IAM user must have, at a minimum, the permissions listed in [IAM permissions for COPY, UNLOAD, and CREATE LIBRARY \(p. 718\)](#).

#### Temporary security credentials

If you are using key-based access control, you can further limit the access users have to your data by using temporary security credentials. Role-based authentication automatically uses temporary credentials.

##### Note

We strongly recommend using [role-based access control \(p. 715\)](#) instead of creating temporary credentials and providing access key ID and secret access key as plain text. Role-based access control automatically uses temporary credentials.

Temporary security credentials provide enhanced security because they have short lifespans and can't be reused after they expire. The access key ID and secret access key generated with the token can't be used without the token, and a user who has these temporary security credentials can access your resources only until the credentials expire.

To grant users temporary access to your resources, you call AWS Security Token Service (AWS STS) API operations. The AWS STS API operations return temporary security credentials consisting of a security token, an access key ID, and a secret access key. You issue the temporary security credentials to the users who need temporary access to your resources. These users can be existing IAM users, or they can be non-

AWS users. For more information about creating temporary security credentials, see [Using Temporary Security Credentials](#) in the IAM User Guide.

You can use either the [ACCESS\\_KEY\\_ID](#) and [SECRET\\_ACCESS\\_KEY](#) (p. 694) parameters together with the [SESSION\\_TOKEN](#) (p. 694) parameter or the [CREDENTIALS](#) (p. 694) parameter. You must also supply the access key ID and secret access key that were provided with the token.

To authenticate using ACCESS\_KEY\_ID, SECRET\_ACCESS\_KEY, and SESSION\_TOKEN, replace `<temporary-access-key-id>`, `<temporary-secret-access-key>`, and `<temporary-token>` as shown following.

```
ACCESS_KEY_ID '<temporary-access-key-id>'
SECRET_ACCESS_KEY '<temporary-secret-access-key>'
SESSION_TOKEN '<temporary-token>;'
```

To authenticate using CREDENTIALS, include `token=<temporary-token>` in the credentials string as shown following.

```
CREDENTIALS
'aws_access_key_id=<temporary-access-key-id>;aws_secret_access_key=<temporary-secret-access-key>;token=<temporary-token>;';
```

The following example shows a COPY command with temporary security credentials.

```
copy table-name
from 's3://objectpath'
access_key_id '<temporary-access-key-id>'
secret_access_key '<temporary-secret-access-key>'
token '<temporary-token>;'
```

The following example loads the LISTING table with temporary credentials and file encryption.

```
copy listing
from 's3://mybucket/data/listings_pipe.txt'
access_key_id '<temporary-access-key-id>'
secret_access_key '<temporary-secret-access-key>'
token '<temporary-token>'
master_symmetric_key '<root-key>'
encrypted;
```

The following example loads the LISTING table using the CREDENTIALS parameter with temporary credentials and file encryption.

```
copy listing
from 's3://mybucket/data/listings_pipe.txt'
credentials
'aws_access_key_id=<temporary-access-key-id>;aws_secret_access_key=<temporary-secret-access-key>;token=<temporary-token>;master_symmetric_key=<root-key>'
encrypted;
```

### Important

The temporary security credentials must be valid for the entire duration of the COPY or UNLOAD operation. If the temporary security credentials expire during the operation, the command fails and the transaction is rolled back. For example, if temporary security credentials expire after 15 minutes and the COPY operation requires one hour, the COPY operation fails before it completes. If you use role-based access, the temporary security credentials are automatically refreshed until the operation completes.

## IAM permissions for COPY, UNLOAD, and CREATE LIBRARY

The IAM role or user referenced by the CREDENTIALS parameter must have, at a minimum, the following permissions:

- For COPY from Amazon S3, permission to LIST the Amazon S3 bucket and GET the Amazon S3 objects that are being loaded, and the manifest file, if one is used.
- For COPY from Amazon S3, Amazon EMR, and remote hosts (SSH) with JSON-formatted data, permission to LIST and GET the JSONPaths file on Amazon S3, if one is used.
- For COPY from DynamoDB, permission to SCAN and DESCRIBE the DynamoDB table that is being loaded.
- For COPY from an Amazon EMR cluster, permission for the `ListInstances` action on the Amazon EMR cluster.
- For UNLOAD to Amazon S3, GET, LIST, and PUT permissions for the Amazon S3 bucket to which the data files are being unloaded.
- For CREATE LIBRARY from Amazon S3, permission to LIST the Amazon S3 bucket and GET the Amazon S3 objects being imported.

### Note

If you receive the error message `S3ServiceException: Access Denied`, when running a COPY, UNLOAD, or CREATE LIBRARY command, your cluster doesn't have proper access permissions for Amazon S3.

You can manage IAM permissions by attaching an IAM policy to an IAM role that is attached to your cluster, to a user, or to the group to which your user belongs. For example, the `AmazonS3ReadOnlyAccess` managed policy grants LIST and GET permissions to Amazon S3 resources. For more information about IAM policies, see [Managing IAM Policies](#) in the *IAM User Guide*.

## Using COPY with Amazon S3 access point aliases

COPY supports Amazon S3 access point aliases. For more information, see [Using a bucket-style alias for your access point](#) in the *Amazon Simple Storage Service User Guide*.

## Loading multibyte data from Amazon S3

If your data includes non-ASCII multibyte characters (such as Chinese or Cyrillic characters), you must load the data to VARCHAR columns. The VARCHAR data type supports four-byte UTF-8 characters, but the CHAR data type only accepts single-byte ASCII characters. You can't load five-byte or longer characters into Amazon Redshift tables. For more information, see [Multibyte characters \(p. 575\)](#).

## Loading a column of the GEOMETRY or GEOGRAPHY data type

You can COPY to GEOMETRY or GEOGRAPHY columns from data in a character-delimited text file, such as a CSV file. The data must be in the hexadecimal form of the well-known binary format (either WKB or EWKB) or the well-known text format (either WKT or EWKT) and fit within the maximum size of a single input row to the COPY command. For more information, see [COPY \(p. 677\)](#).

For information about how to load from a shapefile, see [Loading a shapefile into Amazon Redshift \(p. 228\)](#).

For more information about the GEOMETRY or GEOGRAPHY data type, see [Querying spatial data in Amazon Redshift \(p. 220\)](#).

## Loading the HLLSKETCH data type

You can copy HLL sketches only in sparse or dense format supported by Amazon Redshift. To use the COPY command on HyperLogLog sketches, use the Base64 format for dense HyperLogLog sketches

and the JSON format for sparse HyperLogLog sketches. For more information, see [HyperLogLog functions \(p. 1138\)](#).

The following example imports data from a CSV file into a table using CREATE TABLE and COPY. First, the example creates the table t1 using CREATE TABLE.

```
CREATE TABLE t1 (sketch hllsketch, a bigint);
```

Then it uses COPY to import data from a CSV file into the table t1.

```
COPY t1 FROM s3://mybucket/unload/ IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole' NULL AS 'null' CSV;
```

## Loading a column of the VARBYTE data type

You can load data from a file in CSV, Parquet, and ORC format. For CSV, the data is loaded from a file in hexadecimal representation of the VARBYTE data. You can't load VARBYTE data with the FIXEDWIDTH option. The ADDQUOTES or REMOVEQUOTES option of COPY is not supported. A VARBYTE column can't be used as a partition column.

## Errors when reading multiple files

The COPY command is atomic and transactional. In other words, even when the COPY command reads data from multiple files, the entire process is treated as a single transaction. If COPY encounters an error reading a file, it automatically retries until the process times out (see [statement\\_timeout \(p. 1699\)](#)) or if data can't be download from Amazon S3 for a prolonged period of time (between 15 and 30 minutes), ensuring that each file is loaded only once. If the COPY command fails, the entire transaction is canceled and all changes are rolled back. For more information about handling load errors, see [Troubleshooting data loads \(p. 90\)](#).

After a COPY command is successfully initiated, it doesn't fail if the session terminates, for example when the client disconnects. However, if the COPY command is within a BEGIN ... END transaction block that doesn't complete because the session terminates, the entire transaction, including the COPY, is rolled back. For more information about transactions, see [BEGIN \(p. 669\)](#).

## COPY from JSON format

The JSON data structure is made up of a set of objects or arrays. A JSON *object* begins and ends with braces, and contains an unordered collection of name-value pairs. Each name and value are separated by a colon, and the pairs are separated by commas. The name is a string in double quotation marks. The quotation mark characters must be simple quotation marks (0x22), not slanted or "smart" quotation marks.

A JSON *array* begins and ends with brackets, and contains an ordered collection of values separated by commas. A value can be a string in double quotation marks, a number, a Boolean true or false, null, a JSON object, or an array.

JSON objects and arrays can be nested, enabling a hierarchical data structure. The following example shows a JSON data structure with two valid objects.

```
{
 "id": 1006410,
 "title": "Amazon Redshift Database Developer Guide"
}
{
 "id": 100540,
 "name": "Amazon Simple Storage Service User Guide"
```

```
}
```

The following shows the same data as two JSON arrays.

```
[
 1006410,
 "Amazon Redshift Database Developer Guide"
,
[
 100540,
 "Amazon Simple Storage Service User Guide"
]
```

## COPY options for JSON

You can specify the following options when using COPY with JSON format data:

- 'auto' – COPY automatically loads fields from the JSON file.
- 'auto ignorecase' – COPY automatically loads fields from the JSON file while ignoring the case of field names.
- `s3://jsonpaths_file` – COPY uses a JSONPaths file to parse the JSON source data. A *JSONPaths file* is a text file that contains a single JSON object with the name "jsonpaths" paired with an array of JSONPath expressions. If the name is any string other than "jsonpaths", COPY uses the 'auto' argument instead of using the JSONPaths file.

For examples that show how to load data using 'auto', 'auto ignorecase', or a JSONPaths file, and using either JSON objects or arrays, see [Copy from JSON examples \(p. 734\)](#).

### JSONPath option

In the Amazon Redshift COPY syntax, a JSONPath expression specifies the explicit path to a single name element in a JSON hierarchical data structure, using either bracket notation or dot notation. Amazon Redshift doesn't support any JSONPath elements, such as wildcard characters or filter expressions, that might resolve to an ambiguous path or multiple name elements. As a result, Amazon Redshift can't parse complex, multi-level data structures.

The following is an example of a JSONPaths file with JSONPath expressions using bracket notation. The dollar sign (\$) represents the root-level structure.

```
{
 "jsonpaths": [
 "$['id']",
 "$['store']['book']['title']",
 "$['location'][0]"
]
}
```

In the previous example, `$['location'][0]` references the first element in an array. JSON uses zero-based array indexing. Array indexes must be positive integers (greater than or equal to zero).

The following example shows the previous JSONPaths file using dot notation.

```
{
 "jsonpaths": [
 ".id",
 ".store.book.title",
 ".location[0]"
]
}
```

```
]
}
```

You can't mix bracket notation and dot notation in the jsonpaths array. Brackets can be used in both bracket notation and dot notation to reference an array element.

When using dot notation, the JSONPath expressions can't contain the following characters:

- Single straight quotation mark (' )
- Period, or dot ( . )
- Brackets ( [ ] ) unless used to reference an array element

If the value in the name-value pair referenced by a JSONPath expression is an object or an array, the entire object or array is loaded as a string, including the braces or brackets. For example, suppose that your JSON data contains the following object.

```
{
 "id": 0,
 "guid": "84512477-fa49-456b-b407-581d0d851c3c",
 "isActive": true,
 "tags": [
 "nisi",
 "culpa",
 "ad",
 "amet",
 "voluptate",
 "reprehenderit",
 "veniam"
],
 "friends": [
 {
 "id": 0,
 "name": "Martha Rivera"
 },
 {
 "id": 1,
 "name": "Renaldo"
 }
]
}
```

The JSONPath expression `$['tags']` then returns the following value.

```
["nisi","culpa","ad","amet","voluptate","reprehenderit","veniam"]
```

The JSONPath expression `$['friends'][1]` then returns the following value.

```
{"id": 1,"name": "Renaldo"}
```

Each JSONPath expression in the jsonpaths array corresponds to one column in the Amazon Redshift target table. The order of the jsonpaths array elements must match the order of the columns in the target table or the column list, if a column list is used.

For examples that show how to load data using either the 'auto' argument or a JSONPaths file, and using either JSON objects or arrays, see [Copy from JSON examples \(p. 734\)](#).

For information on how to copy multiple JSON files, see [Using a manifest to specify data files \(p. 71\)](#).

## Escape characters in JSON

COPY loads \n as a newline character and loads \t as a tab character. To load a backslash, escape it with a backslash (\\\).

For example, suppose you have the following JSON in a file named escape.json in the bucket s3://mybucket/json/.

```
{
 "backslash": "This is a backslash: \\",
 "newline": "This sentence\n is on two lines.",
 "tab": "This sentence \t contains a tab."
}
```

Run the following commands to create the ESCAPES table and load the JSON.

```
create table escapes (backslash varchar(25), newline varchar(35), tab varchar(35));

copy escapes from 's3://mybucket/json/escape.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as json 'auto';
```

Query the ESCAPES table to view the results.

```
select * from escapes;

+-----+-----+-----+
| backslash | newline | tab |
+-----+-----+-----+
| This is a backslash: \ | This sentence | This sentence | contains a tab.
| : is on two lines. |
(1 row)
```

## Loss of numeric precision

You might lose precision when loading numbers from data files in JSON format to a column that is defined as a numeric data type. Some floating point values aren't represented exactly in computer systems. As a result, data you copy from a JSON file might not be rounded as you expect. To avoid a loss of precision, we recommend using one of the following alternatives:

- Represent the number as a string by enclosing the value in double quotation characters.
- Use [ROUNDDEC \(p. 710\)](#) to round the number instead of truncating.
- Instead of using JSON or Avro files, use CSV, character-delimited, or fixed-width text files.

## COPY from columnar data formats

COPY can load data from Amazon S3 in the following columnar formats:

- ORC
- Parquet

COPY supports columnar formatted data with the following restrictions:

- The cluster must be in one of the following AWS Regions:
  - US East (N. Virginia) Region (us-east-1)
  - US East (Ohio) Region (us-east-2)

- US West (N. California) Region (us-west-1)
- US West (Oregon) Region (us-west-2)
- Africa (Cape Town) Region (af-south-1)
- Asia Pacific (Hong Kong) Region (ap-east-1)
- Asia Pacific (Melbourne) Region (ap-southeast-4)
- Asia Pacific (Mumbai) Region (ap-south-1)
- Asia Pacific (Osaka) Region (ap-northeast-3)
- Asia Pacific (Seoul) Region (ap-northeast-2)
- Asia Pacific (Singapore) Region (ap-southeast-1)
- Asia Pacific (Sydney) Region (ap-southeast-2)
- Asia Pacific (Tokyo) Region (ap-northeast-1)
- Canada (Central) Region (ca-central-1)
- China (Beijing) Region (cn-north-1)
- China (Ningxia) Region (cn-northwest-1)
- Europe (Frankfurt) Region (eu-central-1)
- Europe (Ireland) Region (eu-west-1)
- Europe (London) Region (eu-west-2)
- Europe (Milan) Region (eu-south-1)
- Europe (Paris) Region (eu-west-3)
- Europe (Stockholm) Region (eu-north-1)
- Middle East (Bahrain) Region (me-south-1)
- Middle East (UAE) Region (me-central-1)
- South America (São Paulo) Region (sa-east-1)
- AWS GovCloud (US-East) (us-gov-east-1)
- AWS GovCloud (US-West) (us-gov-west-1)
- The Amazon S3 bucket must be in the same AWS Region as the Amazon Redshift database.
- To access your Amazon S3 data through a VPC endpoint, set up access using IAM policies and IAM roles as described in [Using Amazon Redshift Spectrum with Enhanced VPC Routing](#) in the *Amazon Redshift Management Guide*.
- COPY doesn't automatically apply compression encodings.
- Only the following COPY parameters are supported:
  - [ACCEPTINVCHARS \(p. 707\)](#) when copying from an ORC or Parquet file.
  - [FILLRECORD \(p. 709\)](#)
  - [FROM \(p. 684\)](#)
  - [IAM\\_ROLE \(p. 694\)](#)
  - [CREDENTIALS \(p. 694\)](#)
  - [STATUPDATE \(p. 713\)](#)
  - [MANIFEST \(p. 686\)](#)
  - [ACCESS\\_KEY\\_ID, SECRET\\_ACCESS\\_KEY, and SESSION\\_TOKEN \(p. 694\)](#)
  - [EXPLICIT\\_IDS \(p. 709\)](#)
- If COPY encounters an error while loading, the command fails. ACCEPTANYDATE and MAXERROR aren't supported for columnar data types.
- Error messages are sent to the SQL client. Some errors are logged in STL\_LOAD\_ERRORS and STL\_ERROR.
- COPY inserts values into the target table's columns in the same order as the columns occur in the columnar data files. The number of columns in the target table and the number of columns in the data file must match.

- If the file you specify for the COPY operation includes one of the following extensions, we decompress the data without the need for adding any parameters:
  - .gz
  - .snappy
  - .bz2
- COPY from the Parquet and ORC file formats uses Redshift Spectrum and the bucket access. To use COPY for these formats, be sure there are no IAM policies blocking the use of presigned URLs. For more information, see [Using Amazon Redshift Spectrum with enhanced VPC routing](#).

## DATEFORMAT and TIMEFORMAT strings

The COPY command uses the DATEFORMAT and TIMEFORMAT options to parse date and time values in your source data. DATEFORMAT and TIMEFORMAT are formatted strings that must match the format of your source data's date and time values. For example, a COPY command loading source data with the date value Jan-01-1999 must include the following DATEFORMAT string:

```
COPY ...
 DATETIME AS 'MON-DD-YYYY'
```

For more information on managing COPY data conversions, see [Data conversion parameters](#). For an example of using TIMEFORMAT, see [Load a timestamp or datestamp \(p. 732\)](#).

DATEFORMAT and TIMEFORMAT strings can contain datetime separators (such as '-', '/', or ':'), as well as the datepart and timepart formats in the following table.

**Note**

If you can't match the format of your date or time values with the following dateparts and timeparts, or if you have date and time values that use formats different from each other, use the 'auto' argument with the DATEFORMAT or TIMEFORMAT parameter. The 'auto' argument recognizes several formats that aren't supported when using a DATEFORMAT or TIMEFORMAT string. For more information, see [Using automatic recognition with DATEFORMAT and TIMEFORMAT \(p. 725\)](#).

| Datepart or timepart | Meaning                                                                                                                                                                                              |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| YY                   | Year without century                                                                                                                                                                                 |
| YYYY                 | Year with century                                                                                                                                                                                    |
| MM                   | Month as a number                                                                                                                                                                                    |
| MON                  | Month as a name (abbreviated name or full name)                                                                                                                                                      |
| DD                   | Day of month as a number                                                                                                                                                                             |
| HH or HH24           | Hour (24-hour clock) <p><b>Note</b></p> <p>In DATETIME format strings for SQL functions, HH is the same as HH12. However, in DATEFORMAT and TIMEFORMAT strings for COPY, HH is the same as HH24.</p> |
| HH12                 | Hour (12-hour clock)                                                                                                                                                                                 |
| MI                   | Minutes                                                                                                                                                                                              |

| Datepart or timepart | Meaning                                |
|----------------------|----------------------------------------|
| SS                   | Seconds                                |
| AM or PM             | Meridian indicator (for 12-hour clock) |

The default date format is YYYY-MM-DD. The default timestamp without time zone (TIMESTAMP) format is YYYY-MM-DD HH:MI:SS. The default timestamp with time zone (TIMESTAMPTZ) format is YYYY-MM-DD HH:MI:SSOF, where OF is the offset from UTC (for example, -8:00. You can't include a time zone specifier (TZ, tz, or OF) in the timeformat\_string. The seconds (SS) field also supports fractional seconds up to a microsecond level of detail. To load TIMESTAMPTZ data that is in a format different from the default format, specify 'auto'.

Following are some sample dates or times you can encounter in your source data, and the corresponding DATEFORMAT or TIMEFORMAT strings for them.

| Example of source data date or time | DATEFORMAT or TIMEFORMAT Syntax     |
|-------------------------------------|-------------------------------------|
| 03/31/2003                          | DATEFORMAT AS 'MM/DD/YYYY'          |
| March 31, 2003                      | DATEFORMAT AS 'MON DD, YYYY'        |
| 03.31.2003 18:45:05                 | TIMEFORMAT AS 'MM.DD.YYYY HH:MI:SS' |
| 03.31.2003 18:45:05.123456          |                                     |

## Using automatic recognition with DATEFORMAT and TIMEFORMAT

If you specify 'auto' as the argument for the DATEFORMAT or TIMEFORMAT parameter, Amazon Redshift will automatically recognize and convert the date format or time format in your source data. The following shows an example.

```
copy favoritemovies from 'dynamodb://ProductCatalog'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
dateformat 'auto';
```

When used with the 'auto' argument for DATEFORMAT and TIMEFORMAT, COPY recognizes and converts the date and time formats listed in the table in [DATEFORMAT and TIMEFORMAT strings \(p. 724\)](#). In addition, the 'auto' argument recognizes the following formats that aren't supported when using a DATEFORMAT and TIMEFORMAT string.

| Format          | Example of Valid Input String |
|-----------------|-------------------------------|
| Julian          | J2451187                      |
| BC              | Jan-08-95 BC                  |
| YYYYMMDD HHMISS | 19960108 040809               |
| YYMMDD HHMISS   | 960108 040809                 |
| YYYY.DDD        | 1996.008                      |

| Format                    | Example of Valid Input String |
|---------------------------|-------------------------------|
| YYYY-MM-DD HH:MI:SS.SSS   | 1996-01-08 04:05:06.789       |
| DD Mon HH:MI:SS YYYY TZ   | 17 Dec 07:37:16 1997 PST      |
| MM/DD/YYYY HH:MI:SS.SS TZ | 12/17/1997 07:37:16.00 PST    |
| YYYY-MM-DD HH:MI:SS+/-TZ  | 1997-12-17 07:37:16-08        |
| DD.MM.YYYY HH:MI:SS TZ    | 12.17.1997 07:37:16.00 PST    |

Automatic recognition doesn't support epochsecs and epochmillisecs.

To test whether a date or timestamp value will be automatically converted, use a `CAST` function to attempt to convert the string to a date or timestamp value. For example, the following commands test the timestamp value '`J2345678 04:05:06.789`':

```
create table formattest (test char(21));
insert into formattest values('J2345678 04:05:06.789');
select test, cast(test as timestamp) as timestamp, cast(test as date) as date from
formattest;

test | timestamp | date
-----+-----+-----+
J2345678 04:05:06.789 1710-02-23 04:05:06 1710-02-23
```

If the source data for a DATE column includes time information, the time component is truncated. If the source data for a TIMESTAMP column omits time information, 00:00:00 is used for the time component.

## COPY examples

### Note

These examples contain line breaks for readability. Do not include line breaks or spaces in your `credentials-args` string.

### Topics

- [Load FAVORITEMOVIES from an DynamoDB table \(p. 727\)](#)
- [Load LISTING from an Amazon S3 bucket \(p. 727\)](#)
- [Load LISTING from an Amazon EMR cluster \(p. 727\)](#)
- [Using a manifest to specify data files \(p. 727\)](#)
- [Load LISTING from a pipe-delimited file \(default delimiter\) \(p. 729\)](#)
- [Load LISTING using columnar data in Parquet format \(p. 729\)](#)
- [Load LISTING using columnar data in ORC format \(p. 729\)](#)
- [Load EVENT with options \(p. 729\)](#)
- [Load VENUE from a fixed-width data file \(p. 730\)](#)
- [Load CATEGORY from a CSV file \(p. 730\)](#)
- [Load VENUE with explicit values for an IDENTITY column \(p. 731\)](#)
- [Load TIME from a pipe-delimited GZIP file \(p. 731\)](#)
- [Load a timestamp or datestamp \(p. 732\)](#)
- [Load data from a file with default values \(p. 732\)](#)
- [COPY data with the ESCAPE option \(p. 734\)](#)
- [Copy from JSON examples \(p. 734\)](#)

- [Copy from Avro examples \(p. 738\)](#)
- [Preparing files for COPY with the ESCAPE option \(p. 740\)](#)
- [Loading a shapefile into Amazon Redshift \(p. 742\)](#)
- [COPY command with the NOLOAD option \(p. 745\)](#)

## Load FAVORITEMOVIES from an DynamoDB table

The AWS SDKs include a simple example of creating a DynamoDB table called *Movies*. (For this example, see [Getting Started with DynamoDB](#).) The following example loads the Amazon Redshift MOVIES table with data from the DynamoDB table. The Amazon Redshift table must already exist in the database.

```
copy favoritemovies from 'dynamodb://Movies'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
readratio 50;
```

## Load LISTING from an Amazon S3 bucket

The following example loads LISTING from an Amazon S3 bucket. The COPY command loads all of the files in the /data/listing/ folder.

```
copy listing
from 's3://mybucket/data/listing/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

## Load LISTING from an Amazon EMR cluster

The following example loads the SALES table with tab-delimited data from lzop-compressed files in an Amazon EMR cluster. COPY loads every file in the myoutput/ folder that begins with part-.

```
copy sales
from 'emr://j-SAMPLE2B500FC/myoutput/part-*'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '\t' lzop;
```

The following example loads the SALES table with JSON formatted data in an Amazon EMR cluster. COPY loads every file in the myoutput/json/ folder.

```
copy sales
from 'emr://j-SAMPLE2B500FC/myoutput/json/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
JSON 's3://mybucket/jsonpaths.txt';
```

## Using a manifest to specify data files

You can use a manifest to ensure that your COPY command loads all of the required files, and only the required files, from Amazon S3. You can also use a manifest when you need to load multiple files from different buckets or files that don't share the same prefix.

For example, suppose that you need to load the following three files: custdata1.txt, custdata2.txt, and custdata3.txt. You could use the following command to load all of the files in mybucket that begin with custdata by specifying a prefix:

```
copy category
from 's3://mybucket/custdata'
```

```
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

If only two of the files exist because of an error, COPY loads only those two files and finishes successfully, resulting in an incomplete data load. If the bucket also contains an unwanted file that happens to use the same prefix, such as a file named `custdata.backup` for example, COPY loads that file as well, resulting in unwanted data being loaded.

To ensure that all of the required files are loaded and to prevent unwanted files from being loaded, you can use a manifest file. The manifest is a JSON-formatted text file that lists the files to be processed by the COPY command. For example, the following manifest loads the three files in the previous example.

```
{
 "entries": [
 {
 "url": "s3://mybucket/custdata.1",
 "mandatory": true
 },
 {
 "url": "s3://mybucket/custdata.2",
 "mandatory": true
 },
 {
 "url": "s3://mybucket/custdata.3",
 "mandatory": true
 }
]
}
```

The optional `mandatory` flag indicates whether COPY should terminate if the file doesn't exist. The default is `false`. Regardless of any mandatory settings, COPY terminates if no files are found. In this example, COPY returns an error if any of the files isn't found. Unwanted files that might have been picked up if you specified only a key prefix, such as `custdata.backup`, are ignored, because they aren't on the manifest.

When loading from data files in ORC or Parquet format, a `meta` field is required, as shown in the following example.

```
{
 "entries": [
 {
 "url": "s3://mybucket-alpha/orc/2013-10-04-custdata",
 "mandatory": true,
 "meta": {
 "content_length": 99
 }
 },
 {
 "url": "s3://mybucket-beta/orc/2013-10-05-custdata",
 "mandatory": true,
 "meta": {
 "content_length": 99
 }
 }
]
}
```

The following example uses a manifest named `cust.manifest`.

```
copy customer
from 's3://mybucket/cust.manifest'
```

```
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as orc
manifest;
```

You can use a manifest to load files from different buckets or files that don't share the same prefix. The following example shows the JSON to load data with files whose names begin with a date stamp.

```
{
 "entries": [
 {"url": "s3://mybucket/2013-10-04-custdata.txt", "mandatory": true},
 {"url": "s3://mybucket/2013-10-05-custdata.txt", "mandatory": true},
 {"url": "s3://mybucket/2013-10-06-custdata.txt", "mandatory": true},
 {"url": "s3://mybucket/2013-10-07-custdata.txt", "mandatory": true}
]
}
```

The manifest can list files that are in different buckets, as long as the buckets are in the same AWS Region as the cluster.

```
{
 "entries": [
 {"url": "s3://mybucket-alpha/custdata1.txt", "mandatory": false},
 {"url": "s3://mybucket-beta/custdata1.txt", "mandatory": false},
 {"url": "s3://mybucket-beta/custdata2.txt", "mandatory": false}
]
}
```

## Load LISTING from a pipe-delimited file (default delimiter)

The following example is a very simple case in which no options are specified and the input file contains the default delimiter, a pipe character ('|').

```
copy listing
from 's3://mybucket/data/listings_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

## Load LISTING using columnar data in Parquet format

The following example loads data from a folder on Amazon S3 named parquet.

```
copy listing
from 's3://mybucket/data/listings/parquet/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as parquet;
```

## Load LISTING using columnar data in ORC format

The following example loads data from a folder on Amazon S3 named orc.

```
copy listing
from 's3://mybucket/data/listings/orc/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as orc;
```

## Load EVENT with options

The following example loads pipe-delimited data into the EVENT table and applies the following rules:

- If pairs of quotation marks are used to surround any character strings, they are removed.
- Both empty strings and strings that contain blanks are loaded as NULL values.
- The load fails if more than 5 errors are returned.
- Timestamp values must comply with the specified format; for example, a valid timestamp is 2008-09-26 05:43:12.

```
copy event
from 's3://mybucket/data/allevents_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
removequotes
emptyasnull
blanksasnull
maxerror 5
delimiter '|'
timeformat 'YYYY-MM-DD HH:MI:SS';
```

## Load VENUE from a fixed-width data file

```
copy venue
from 's3://mybucket/data/venue_fw.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
fixedwidth 'venueid:3,venuename:25,venuecity:12,venuestate:2,venueseats:6';
```

The preceding example assumes a data file formatted in the same way as the sample data shown. In the sample following, spaces act as placeholders so that all of the columns are the same width as noted in the specification:

|   |                           |             |         |
|---|---------------------------|-------------|---------|
| 1 | Toyota Park               | Bridgeview  | IL0     |
| 2 | Columbus Crew Stadium     | Columbus    | OH0     |
| 3 | RFK Stadium               | Washington  | DC0     |
| 4 | CommunityAmerica Ballpark | Kansas City | KS0     |
| 5 | Gillette Stadium          | Foxborough  | MA68756 |

## Load CATEGORY from a CSV file

Suppose you want to load the CATEGORY with the values shown in the following table.

| catid | catgroup | catname   | catdesc                                    |
|-------|----------|-----------|--------------------------------------------|
| 12    | Shows    | Musicals  | Musical theatre                            |
| 13    | Shows    | Plays     | All "non-musical" theatre                  |
| 14    | Shows    | Opera     | All opera, light, and "rock" opera         |
| 15    | Concerts | Classical | All symphony, concerto, and choir concerts |

The following example shows the contents of a text file with the field values separated by commas.

```
12,Shows,Musicals,Musical theatre
13,Shows,Plays,All "non-musical" theatre
14,Shows,Opera,All opera, light, and "rock" opera
15,Concerts,Classical,All symphony, concerto, and choir concerts
```

If you load the file using the DELIMITER parameter to specify comma-delimited input, the COPY command fails because some input fields contain commas. You can avoid that problem by using the CSV parameter and enclosing the fields that contain commas in quotation mark characters. If the quotation mark character appears within a quoted string, you need to escape it by doubling the quotation mark character. The default quotation mark character is a double quotation mark, so you need to escape each double quotation mark with an additional double quotation mark. Your new input file looks something like this.

```
12,Shows,Musicals,Musical theatre
13,Shows,Plays,"All ""non-musical"" theatre"
14,Shows,Opera,"All opera, light, and ""rock"" opera"
15,Concerts,Classical,"All symphony, concerto, and choir concerts"
```

Assuming the file name is category\_csv.txt, you can load the file by using the following COPY command:

```
copy category
from 's3://mybucket/data/category_csv.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
csv;
```

Alternatively, to avoid the need to escape the double quotation marks in your input, you can specify a different quotation mark character by using the QUOTE AS parameter. For example, the following version of category\_csv.txt uses '%' as the quotation mark character.

```
12,Shows,Musicals,Musical theatre
13,Shows,Plays,%All "non-musical" theatre%
14,Shows,Opera,%All opera, light, and "rock" opera%
15,Concerts,Classical,%All symphony, concerto, and choir concerts%
```

The following COPY command uses QUOTE AS to load category\_csv.txt:

```
copy category
from 's3://mybucket/data/category_csv.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
csv quote as '%';
```

## Load VENUE with explicit values for an IDENTITY column

The following example assumes that when the VENUE table was created that at least one column (such as the venueid column) was specified to be an IDENTITY column. This command overrides the default IDENTITY behavior of autogenerated values for an IDENTITY column and instead loads the explicit values from the venue.txt file. Amazon Redshift does not check if duplicate IDENTITY values are loaded into the table when using the EXPLICIT\_IDS option.

```
copy venue
from 's3://mybucket/data/venue.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
explicit_ids;
```

## Load TIME from a pipe-delimited GZIP file

The following example loads the TIME table from a pipe-delimited GZIP file:

```
copy time
from 's3://mybucket/data/timerows.gz'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
```

```
gzip
delimiter '|';
```

## Load a timestamp or datestamp

The following example loads data with a formatted timestamp.

### Note

The TIMEFORMAT of HH:MI:SS can also support fractional seconds beyond the SS to a microsecond level of detail. The file time.txt used in this example contains one row, 2009-01-12 14:15:57.119568.

```
copy timestamp1
from 's3://mybucket/data/time.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
timeformat 'YYYY-MM-DD HH:MI:SS';
```

The result of this copy is as follows:

```
select * from timestamp1;
c1

2009-01-12 14:15:57.119568
(1 row)
```

## Load data from a file with default values

The following example uses a variation of the VENUE table in the TICKIT database. Consider a VENUE\_NEW table defined with the following statement:

```
create table venue_new(
venueid smallint not null,
venuename varchar(100) not null,
venuecity varchar(30),
venuestate char(2),
venueseats integer not null default '1000');
```

Consider a venue\_noseats.txt data file that contains no values for the VENUESEATS column, as shown in the following example:

```
1|Toyota Park|Bridgeview|IL|
2|Columbus Crew Stadium|Columbus|OH|
3|RFK Stadium|Washington|DC|
4|CommunityAmerica Ballpark|Kansas City|KS|
5|Gillette Stadium|Foxborough|MA|
6|New York Giants Stadium|East Rutherford|NJ|
7|BMO Field|Toronto|ON|
8|The Home Depot Center|Carson|CA|
9|Dick's Sporting Goods Park|Commerce City|CO|
10|Pizza Hut Park|Frisco|TX|
```

The following COPY statement will successfully load the table from the file and apply the DEFAULT value ('1000') to the omitted column:

```
copy venue_new(venueid, venuename, venuecity, venuestate)
from 's3://mybucket/data/venue_noseats.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

Now view the loaded table:

```
select * from venue_new order by venueid;
venueid | venuename | venuecity | venuestate | venueseats
-----+-----+-----+-----+-----+
1 | Toyota Park | Bridgeview | IL | 1000
2 | Columbus Crew Stadium | Columbus | OH | 1000
3 | RFK Stadium | Washington | DC | 1000
4 | CommunityAmerica Ballpark | Kansas City | KS | 1000
5 | Gillette Stadium | Foxborough | MA | 1000
6 | New York Giants Stadium | East Rutherford | NJ | 1000
7 | BMO Field | Toronto | ON | 1000
8 | The Home Depot Center | Carson | CA | 1000
9 | Dick's Sporting Goods Park | Commerce City | CO | 1000
10 | Pizza Hut Park | Frisco | TX | 1000
(10 rows)
```

For the following example, in addition to assuming that no VENUESEATS data is included in the file, also assume that no VENUENAME data is included:

```
1||Bridgeview|IL|
2||Columbus|OH|
3||Washington|DC|
4||Kansas City|KS|
5||Foxborough|MA|
6||East Rutherford|NJ|
7||Toronto|ON|
8||Carson|CA|
9||Commerce City|CO|
10||Frisco|TX|
```

Using the same table definition, the following COPY statement fails because no DEFAULT value was specified for VENUENAME, and VENUENAME is a NOT NULL column:

```
copy venue(venueid, venuecity, venuestate)
from 's3://mybucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

Now consider a variation of the VENUE table that uses an IDENTITY column:

```
create table venue_identity(
venueid int identity(1,1),
venuename varchar(100) not null,
venuecity varchar(30),
venuestate char(2),
venueseats integer not null default '1000');
```

As with the previous example, assume that the VENUESEATS column has no corresponding values in the source file. The following COPY statement successfully loads the table, including the predefined IDENTITY data values instead of autogenerated those values:

```
copy venue(venueid, venuename, venuecity, venuestate)
from 's3://mybucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' explicit_ids;
```

This statement fails because it doesn't include the IDENTITY column (VENUEID is missing from the column list) yet includes an EXPLICIT\_IDS parameter:

```
copy venue(venuename, venuecity, venuestate)
from 's3://mybucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' explicit_ids;
```

This statement fails because it doesn't include an EXPLICIT\_IDS parameter:

```
copy venue(venueid, venuename, venuecity, venuestate)
from 's3://mybucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

## COPY data with the ESCAPE option

The following example shows how to load characters that match the delimiter character (in this case, the pipe character). In the input file, make sure that all of the pipe characters (|) that you want to load are escaped with the backslash character (\). Then load the file with the ESCAPE parameter.

```
$ more redshiftinfo.txt
1|public\|event\|dwuser
2|public\|sales\|dwuser

create table redshiftinfo(info_id int, table_info varchar(50));

copy redshiftinfo from 's3://mybucket/data/redshiftinfo.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' escape;

select * from redshiftinfo order by 1;
info_id | table_info
-----+-----
1 | public|event|dwuser
2 | public|sales|dwuser
(2 rows)
```

Without the ESCAPE parameter, this COPY command fails with an Extra column(s) found error.

### Important

If you load your data using a COPY with the ESCAPE parameter, you must also specify the ESCAPE parameter with your UNLOAD command to generate the reciprocal output file.

Similarly, if you UNLOAD using the ESCAPE parameter, you need to use ESCAPE when you COPY the same data.

## Copy from JSON examples

In the following examples, you load the CATEGORY table with the following data.

| CATID | CATGROUP | CATNAME   | CATDESC                                    |
|-------|----------|-----------|--------------------------------------------|
| 1     | Sports   | MLB       | Major League Baseball                      |
| 2     | Sports   | NHL       | National Hockey League                     |
| 3     | Sports   | NFL       | National Football League                   |
| 4     | Sports   | NBA       | National Basketball Association            |
| 5     | Concerts | Classical | All symphony, concerto, and choir concerts |

## Topics

- [Load from JSON data using the 'auto' option \(p. 735\)](#)
- [Load from JSON data using the 'auto ignorecase' option \(p. 735\)](#)
- [Load from JSON data using a JSONPaths file \(p. 736\)](#)
- [Load from JSON arrays using a JSONPaths file \(p. 737\)](#)

### Load from JSON data using the 'auto' option

To load from JSON data using the 'auto' option, the JSON data must consist of a set of objects. The key names must match the column names, but the order doesn't matter. The following shows the contents of a file named category\_object\_auto.json.

```
{
 "catdesc": "Major League Baseball",
 "catid": 1,
 "catgroup": "Sports",
 "catname": "MLB"
}
{
 "catgroup": "Sports",
 "catid": 2,
 "catname": "NHL",
 "catdesc": "National Hockey League"
}{
 "catid": 3,
 "catname": "NFL",
 "catgroup": "Sports",
 "catdesc": "National Football League"
}
{
 "bogus": "Bogus Sports LLC",
 "catid": 4,
 "catgroup": "Sports",
 "catname": "NBA",
 "catdesc": "National Basketball Association"
}
{
 "catid": 5,
 "catgroup": "Shows",
 "catname": "Musicals",
 "catdesc": "All symphony, concerto, and choir concerts"
}
```

To load from the JSON data file in the previous example, run the following COPY command.

```
copy category
from 's3://mybucket/category_object_auto.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 'auto';
```

### Load from JSON data using the 'auto ignorecase' option

To load from JSON data using the 'auto ignorecase' option, the JSON data must consist of a set of objects. The case of the key names doesn't have to match the column names and the order doesn't matter. The following shows the contents of a file named category\_object\_auto-ignorecase.json.

```
{
```

```

 "CatDesc": "Major League Baseball",
 "CatID": 1,
 "CatGroup": "Sports",
 "CatName": "MLB"
 }
{
 "CatGroup": "Sports",
 "CatID": 2,
 "CatName": "NHL",
 "CatDesc": "National Hockey League"
}
{
 "CatID": 3,
 "CatName": "NFL",
 "CatGroup": "Sports",
 "CatDesc": "National Football League"
}
{
 "bogus": "Bogus Sports LLC",
 "CatID": 4,
 "CatGroup": "Sports",
 "CatName": "NBA",
 "CatDesc": "National Basketball Association"
}
{
 "CatID": 5,
 "CatGroup": "Shows",
 "CatName": "Musicals",
 "CatDesc": "All symphony, concerto, and choir concerts"
}

```

To load from the JSON data file in the previous example, run the following COPY command.

```

copy category
from 's3://mybucket/category_object_auto ignorecase.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 'auto ignorecase';

```

### [Load from JSON data using a JSONPaths file](#)

If the JSON data objects don't correspond directly to column names, you can use a JSONPaths file to map the JSON elements to columns. The order doesn't matter in the JSON source data, but the order of the JSONPaths file expressions must match the column order. Suppose that you have the following data file, named `category_object_paths.json`.

```

{
 "one": 1,
 "two": "Sports",
 "three": "MLB",
 "four": "Major League Baseball"
}
{
 "three": "NHL",
 "four": "National Hockey League",
 "one": 2,
 "two": "Sports"
}
{
 "two": "Sports",
 "three": "NFL",
 "one": 3,
 "four": "National Football League"
}

```

```

 "one": 4,
 "two": "Sports",
 "three": "NBA",
 "four": "National Basketball Association"
 }
{
 "one": 6,
 "two": "Shows",
 "three": "Musicals",
 "four": "All symphony, concerto, and choir concerts"
}

```

The following JSONPaths file, named `category_jsonpath.json`, maps the source data to the table columns.

```

{
 "jsonpaths": [
 "$['one']",
 "$['two']",
 "$['three']",
 "$['four']"
]
}

```

To load from the JSON data file in the previous example, run the following COPY command.

```

copy category
from 's3://mybucket/category_object_paths.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 's3://mybucket/category_jsonpath.json';

```

#### Load from JSON arrays using a JSONPaths file

To load from JSON data that consists of a set of arrays, you must use a JSONPaths file to map the array elements to columns. Suppose that you have the following data file, named `category_array_data.json`.

```

[1,"Sports","MLB","Major League Baseball"]
[2,"Sports","NHL","National Hockey League"]
[3,"Sports","NFL","National Football League"]
[4,"Sports","NBA","National Basketball Association"]
[5,"Concerts","Classical","All symphony, concerto, and choir concerts"]

```

The following JSONPaths file, named `category_array_jsonpath.json`, maps the source data to the table columns.

```

{
 "jsonpaths": [
 "$[0]",
 "$[1]",
 "$[2]",
 "$[3]"
]
}

```

To load from the JSON data file in the previous example, run the following COPY command.

```

copy category

```

```
from 's3://mybucket/category_array_data.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 's3://mybucket/category_array_jsonpath.json';
```

## Copy from Avro examples

In the following examples, you load the CATEGORY table with the following data.

| CATID | CATGROUP | CATNAME   | CATDESC                                    |
|-------|----------|-----------|--------------------------------------------|
| 1     | Sports   | MLB       | Major League Baseball                      |
| 2     | Sports   | NHL       | National Hockey League                     |
| 3     | Sports   | NFL       | National Football League                   |
| 4     | Sports   | NBA       | National Basketball Association            |
| 5     | Concerts | Classical | All symphony, concerto, and choir concerts |

### Topics

- [Load from Avro data using the 'auto' option \(p. 738\)](#)
- [Load from Avro data using the 'auto ignorecase' option \(p. 739\)](#)
- [Load from Avro data using a JSONPaths file \(p. 740\)](#)

### Load from Avro data using the 'auto' option

To load from Avro data using the 'auto' argument, field names in the Avro schema must match the column names. When using the 'auto' argument, order doesn't matter. The following shows the schema for a file named category\_auto.avro.

```
{
 "name": "category",
 "type": "record",
 "fields": [
 {"name": "catid", "type": "int"},
 {"name": "catdesc", "type": "string"},
 {"name": "catname", "type": "string"},
 {"name": "catgroup", "type": "string"},
]
}
```

The data in an Avro file is in binary format, so it isn't human-readable. The following shows a JSON representation of the data in the category\_auto.avro file.

```
{
 "catid": 1,
 "catdesc": "Major League Baseball",
 "catname": "MLB",
 "catgroup": "Sports"
}
{
 "catid": 2,
 "catdesc": "National Hockey League",
 "catname": "NHL",
 "catgroup": "Sports"
}
```

```
{
 "catid": 3,
 "catdesc": "National Basketball Association",
 "catname": "NBA",
 "catgroup": "Sports"
}
{
 "catid": 4,
 "catdesc": "All symphony, concerto, and choir concerts",
 "catname": "Classical",
 "catgroup": "Concerts"
}
```

To load from the Avro data file in the previous example, run the following COPY command.

```
copy category
from 's3://mybucket/category_auto.avro'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as avro 'auto';
```

#### [Load from Avro data using the 'auto ignorecase' option](#)

To load from Avro data using the 'auto ignorecase' argument, the case of the field names in the Avro schema does not have to match the case of column names. When using the 'auto ignorecase' argument, order doesn't matter. The following shows the schema for a file named `category_auto-ignorecase.avro`.

```
{
 "name": "category",
 "type": "record",
 "fields": [
 {"name": "CatID", "type": "int"},
 {"name": "CatDesc", "type": "string"},
 {"name": "CatName", "type": "string"},
 {"name": "CatGroup", "type": "string"},
]
}
```

The data in an Avro file is in binary format, so it isn't human-readable. The following shows a JSON representation of the data in the `category_auto-ignorecase.avro` file.

```
{
 "CatID": 1,
 "CatDesc": "Major League Baseball",
 "CatName": "MLB",
 "CatGroup": "Sports"
}
{
 "CatID": 2,
 "CatDesc": "National Hockey League",
 "CatName": "NHL",
 "CatGroup": "Sports"
}
{
 "CatID": 3,
 "CatDesc": "National Basketball Association",
 "CatName": "NBA",
 "CatGroup": "Sports"
}
{
 "CatID": 4,
 "CatDesc": "All symphony, concerto, and choir concerts",
```

```
 "CatName": "Classical",
 "CatGroup": "Concerts"
}
```

To load from the Avro data file in the previous example, run the following COPY command.

```
copy category
from 's3://mybucket/category_auto-ignorecase.avro'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as avro 'auto ignorecase';
```

### Load from Avro data using a JSONPaths file

If the field names in the Avro schema don't correspond directly to column names, you can use a JSONPaths file to map the schema elements to columns. The order of the JSONPaths file expressions must match the column order.

Suppose that you have a data file named `category_paths.avro` that contains the same data as in the previous example, but with the following schema.

```
{
 "name": "category",
 "type": "record",
 "fields": [
 {"name": "id", "type": "int"},
 {"name": "desc", "type": "string"},
 {"name": "name", "type": "string"},
 {"name": "group", "type": "string"},
 {"name": "region", "type": "string"}
]
}
```

The following JSONPaths file, named `category_path.avropath`, maps the source data to the table columns.

```
{
 "jsonpaths": [
 "$['id']",
 "$['group']",
 "$['name']",
 "$['desc']"
]
}
```

To load from the Avro data file in the previous example, run the following COPY command.

```
copy category
from 's3://mybucket/category_object_paths.avro'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format avro 's3://mybucket/category_path.avropath ';
```

### Preparing files for COPY with the ESCAPE option

The following example describes how you might prepare data to "escape" newline characters before importing the data into an Amazon Redshift table using the COPY command with the ESCAPE parameter. Without preparing the data to delimit the newline characters, Amazon Redshift returns load errors when you run the COPY command, because the newline character is normally used as a record separator.

For example, consider a file or a column in an external table that you want to copy into an Amazon Redshift table. If the file or column contains XML-formatted content or similar data, you need to make sure that all of the newline characters (\n) that are part of the content are escaped with the backslash character (\).

A file or table containing embedded newlines characters provides a relatively easy pattern to match. Each embedded newline character most likely always follows a > character with potentially some white space characters (' ' or tab) in between, as you can see in the following example of a text file named nlTest1.txt.

```
$ cat nlTest1.txt
<xml start>
<newline characters provide>
<line breaks at the end of each>
<line in content>
</xml>|1000
<xml>
</xml>|2000
```

With the following example, you can run a text-processing utility to pre-process the source file and insert escape characters where needed. (The | character is intended to be used as delimiter to separate column data when copied into an Amazon Redshift table.)

```
$ sed -e ':a;N;$!ba;s/>[[[:space:]]*\n/>\\n/g' nlTest1.txt > nlTest2.txt
```

Similarly, you can use Perl to perform a similar operation:

```
cat nlTest1.txt | perl -p -e 's/>\s*\n/>\\n/g' > nlTest2.txt
```

To accommodate loading the data from the nlTest2.txt file into Amazon Redshift, we created a two-column table in Amazon Redshift. The first column c1, is a character column that holds XML-formatted content from the nlTest2.txt file. The second column c2 holds integer values loaded from the same file.

After running the sed command, you can correctly load data from the nlTest2.txt file into an Amazon Redshift table using the ESCAPE parameter.

**Note**

When you include the ESCAPE parameter with the COPY command, it escapes a number of special characters that include the backslash character (including newline).

```
copy t2 from 's3://mybucket/data/nlTest2.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
escape
delimiter as '|';

select * from t2 order by 2;

c1 | c2
-----+-----
<xml start>
<newline characters provide>
<line breaks at the end of each>
<line in content>
</xml>
| 1000
<xml>
</xml> | 2000
(2 rows)
```

You can prepare data files exported from external databases in a similar way. For example, with an Oracle database, you can use the REPLACE function on each affected column in a table that you want to copy into Amazon Redshift.

```
SELECT c1, REPLACE(c2, '\n', '\n') as c2 from my_table_with_xml
```

In addition, many database export and extract, transform, load (ETL) tools that routinely process large amounts of data provide options to specify escape and delimiter characters.

## Loading a shapefile into Amazon Redshift

The following examples demonstrate how to load an Esri shapefile using COPY. For more information about loading shapefiles, see [Loading a shapefile into Amazon Redshift \(p. 228\)](#).

### Loading a shapefile

The following steps show how to ingest OpenStreetMap data from Amazon S3 using the COPY command. This example assumes that the Norway shapefile archive from [the download site of Geofabrik](#) has been uploaded to a private Amazon S3 bucket in your AWS Region. The .shp, .shx, and .dbf files must share the same Amazon S3 prefix and file name.

### Ingesting data without simplification

The following commands create tables and ingest data that can fit in the maximum geometry size without any simplification. Open the gis\_osm\_natural\_free\_1.shp in your preferred GIS software and inspect the columns in this layer. By default, either IDENTITY or GEOMETRY columns are first. When a GEOMETRY column is first, you can create the table as shown following.

```
CREATE TABLE norway_natural (
 wkb_geometry GEOMETRY,
 osm_id BIGINT,
 code INT,
 fclass VARCHAR,
 name VARCHAR);
```

Or, when an IDENTITY column is first, you can create the table as shown following.

```
CREATE TABLE norway_natural_with_id (
 fid INT IDENTITY(1,1),
 wkb_geometry GEOMETRY,
 osm_id BIGINT,
 code INT,
 fclass VARCHAR,
 name VARCHAR);
```

Now you can ingest the data using COPY.

```
COPY norway_natural FROM 's3://bucket_name/shapefiles/norway/gis_osm_natural_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO: Load into table 'norway_natural' completed, 83891 record(s) loaded successfully
```

Or you can ingest the data as shown following.

```
COPY norway_natural_with_id FROM 's3://bucket_name/shapefiles/norway/
gis_osm_natural_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
```

```
INFO: Load into table 'norway_natural_with_id' completed, 83891 record(s) loaded successfully.
```

### Ingesting data with simplification

The following commands create a table and try to ingest data that can't fit in the maximum geometry size without any simplification. Inspect the gis\_osm\_water\_a\_free\_1.shp shapefile and create the appropriate table as shown following.

```
CREATE TABLE norway_water (
 wkb_geometry GEOMETRY,
 osm_id BIGINT,
 code INT,
 fclass VARCHAR,
 name VARCHAR);
```

When the COPY command runs, it results in an error.

```
COPY norway_water FROM 's3://bucket_name/shapefiles/norway/gis_osm_water_a_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
ERROR: Load into table 'norway_water' failed. Check 'stl_load_errors' system table for details.
```

Querying STL\_LOAD\_ERRORS shows that the geometry is too large.

```
SELECT line_number, btrim(colname), btrim(err_reason) FROM stl_load_errors WHERE query =
pg_last_copy_id();
line_number | btrim | btrim
-----+-----+
+-----+
1184705 | wkb_geometry | Geometry size: 1513736 is larger than maximum supported size:
1048447
```

To overcome this, the SIMPLIFY AUTO parameter is added to the COPY command to simplify geometries.

```
COPY norway_water FROM 's3://bucket_name/shapefiles/norway/gis_osm_water_a_free_1.shp'
FORMAT SHAPEFILE
SIMPLIFY AUTO
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';

INFO: Load into table 'norway_water' completed, 1989196 record(s) loaded successfully.
```

To view the rows and geometries that were simplified, query SVL\_SPATIAL\_SIMPLIFY.

```
SELECT * FROM svl_spatial_simplify WHERE query = pg_last_copy_id();
query | line_number | maximum_tolerance | initial_size | simplified | final_size | final_tolerance
-----+-----+-----+-----+-----+-----+
20 | 1184704 | -1 | 1513736 | t | 1008808 |
1.276386653895e-05
20 | 1664115 | -1 | 1233456 | t | 1023584 |
6.11707814796635e-06
```

Using SIMPLIFY AUTO *max\_tolerance* with the tolerance lower than the automatically calculated ones probably results in an ingestion error. In this case, use MAXERROR to ignore errors.

```
COPY norway_water FROM 's3://bucket_name/shapefiles/norway/gis_osm_water_a_free_1.shp'
FORMAT SHAPEFILE
SIMPLIFY AUTO 1.1E-05
MAXERROR 2
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';

INFO: Load into table 'norway_water' completed, 1989195 record(s) loaded successfully.
INFO: Load into table 'norway_water' completed, 1 record(s) could not be loaded. Check
'st_load_errors' system table for details.
```

Query SVL\_SPATIAL\_SIMPLIFY again to identify the record that COPY didn't manage to load.

| 29      | 1184704 | 1.1e-05 | 1513736   f |  | 0      |  |
|---------|---------|---------|-------------|--|--------|--|
| 0       |         |         |             |  |        |  |
| 29      | 1664115 | 1.1e-05 | 1233456   t |  | 794432 |  |
| 1.1e-05 |         |         |             |  |        |  |

In this example, the first record didn't manage to fit, so the simplified column is showing false. The second record was loaded within the given tolerance. However, the final size is larger than using the automatically calculated tolerance without specifying the maximum tolerance.

### Loading from a compressed shapefile

Amazon Redshift COPY supports ingesting data from a compressed shapefile. All shapefile components must have the same Amazon S3 prefix and the same compression suffix. As an example, suppose that you want to load the data from the previous example. In this case, the files gis\_osm\_water\_a\_free\_1.shp.gz, gis\_osm\_water\_a\_free\_1.dbf.gz, and gis\_osm\_water\_a\_free\_1.shx.gz must share the same Amazon S3 directory. The COPY command requires the GZIP option, and the FROM clause must specify the correct compressed file, as shown following.

```
COPY norway_natural FROM 's3://bucket_name/shapefiles/norway/compressed/
gis_osm_natural_free_1.shp.gz'
FORMAT SHAPEFILE
GZIP
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO: Load into table 'norway_natural' completed, 83891 record(s) loaded successfully.
```

### Loading data into a table with a different column order

If you have a table that doesn't have GEOMETRY as the first column, you can use column mapping to map columns to the target table. For example, create a table with osm\_id specified as a first column.

```
CREATE TABLE norway_natural_order (
 osm_id BIGINT,
 wkb_geometry GEOMETRY,
 code INT,
 fclass VARCHAR,
 name VARCHAR);
```

Then ingest a shapefile using column mapping.

```
COPY norway_natural_order(wkb_geometry, osm_id, code, fclass, name)
```

```
FROM 's3://bucket_name/shapefiles/norway/gis_osm_natural_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO: Load into table 'norway_natural_order' completed, 83891 record(s) loaded
successfully.
```

### Loading data into a table with a geography column

If you have a table that has a `GEOGRAPHY` column, you first ingest into a `GEOMETRY` column and then cast the objects to `GEOGRAPHY` objects. For example, after you copy your shapefile into a `GEOMETRY` column, alter the table to add a column of the `GEOGRAPHY` data type.

```
ALTER TABLE norway_natural ADD COLUMN wkb_geography GEOGRAPHY;
```

Then convert geometries to geographies.

```
UPDATE norway_natural SET wkb_geography = wkb_geometry::geography;
```

Optionally, you can drop the `GEOMETRY` column.

```
ALTER TABLE norway_natural DROP COLUMN wkb_geometry;
```

### COPY command with the NOLOAD option

To validate data files before you actually load the data, use the `NOLOAD` option with the `COPY` command. Amazon Redshift parses the input file and displays any errors that occur. The following example uses the `NOLOAD` option and no rows are actually loaded into the table.

```
COPY public.zipcode1
FROM 's3://mybucket/mydata/zipcode.csv'
DELIMITER ';'
IGNOREHEADER 1 REGION 'us-east-1'
NOLOAD
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/myRedshiftRole';

Warnings:
Load into table 'zipcode1' completed, 0 record(s) loaded successfully.
```

## CREATE DATABASE

Creates a new database.

To create a database, you must be a superuser or have the `CREATEDB` privilege.

You can't run `CREATE DATABASE` within a transaction block (`BEGIN ... END`). For more information about transactions, see [Serializable isolation \(p. 119\)](#).

### Syntax

```
CREATE DATABASE database_name [WITH]
[OWNER [=] db_owner]
[CONNECTION LIMIT { limit | UNLIMITED }]
[COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }]
```

```
[ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }]
FROM [DATASHARE datashare_name | ARN '<arn>'] OF [ACCOUNT account_id] NAMESPACE
namespace_guid
[WITH DATA CATALOG SCHEMA '<schema>' | WITH NO DATA CATALOG SCHEMA]
```

## Parameters

*database\_name*

Name of the new database. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

WITH

Optional keyword.

OWNER

Specifies a database owner.

=

Optional character.

*db\_owner*

Username for the database owner.

CONNECTION LIMIT { *limit* | UNLIMITED }

The maximum number of database connections users are permitted to have open concurrently. The limit isn't enforced for superusers. Use the UNLIMITED keyword to permit the maximum number of concurrent connections. A limit on the number of connections for each user might also apply. For more information, see [CREATE USER \(p. 846\)](#). The default is UNLIMITED. To view current connections, query the [STV\\_SESSIONS \(p. 1597\)](#) system view.

**Note**

If both user and database connection limits apply, an unused connection slot must be available that is within both limits when a user attempts to connect.

COLLATE { CASE\_SENSITIVE | CASE\_INSENSITIVE }

A clause that specifies whether string search or comparison is CASE\_SENSITIVE or CASE\_INSENSITIVE. The default is CASE\_SENSITIVE.

ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }

A clause that specifies the isolation level used when queries run against a database.

- SERIALIZABLE isolation – provides full serializability for concurrent transactions. This is the default for a database created in a provisioned cluster. For more information, see [Serializable isolation \(p. 119\)](#).
- SNAPSHOT isolation – provides an isolation level with protection against update and delete conflicts. This is the default for a database created in a serverless namespace.

You can view which concurrency model your database is running as follows:

- Query the STV\_DB\_ISOLATION\_LEVEL catalog view. For more information, see [STV\\_DB\\_ISOLATION\\_LEVEL \(p. 1582\)](#).

```
SELECT * FROM stv_db_isolation_level;
```

- Query the PG\_DATABASE\_INFO view.

```
SELECT datname, datconfig FROM pg_database_info;
```

The isolation level per database appears next to the key `concurrency_model`. A value of 1 denotes **SNAPSHOT**. A value of 2 denotes **SERIALIZABLE**.

In Amazon Redshift databases, both **SERIALIZABLE** and **SNAPSHOT** isolation are types of serializable isolation levels. That is, dirty reads, non-repeatable reads, and phantom reads are prevented according to the SQL standard. Both isolation levels guarantee that a transaction operates on a snapshot of data as it exists when the transaction begins, and that no other transaction can change that snapshot. However, **SNAPSHOT** isolation doesn't provide full serializability, because it doesn't prevent write skew inserts and updates on different table rows.

The following scenario illustrates write skew updates using the **SNAPSHOT** isolation level. A table named `Numbers` contains a column named `digits` that contains 0 and 1 values. Each user's `UPDATE` statement doesn't overlap the other user. However, the 0 and 1 values are swapped. The SQL they run follows this timeline with these results:

| Time                | User 1 action                                                                                                                                                                                                               | User 2 action                                                                                                                                                                                                               |                     |       |   |   |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|-------|---|---|
| 1                   | <code>BEGIN;</code>                                                                                                                                                                                                         |                                                                                                                                                                                                                             |                     |       |   |   |
| 2                   |                                                                                                                                                                                                                             | <code>BEGIN;</code>                                                                                                                                                                                                         |                     |       |   |   |
| 3                   | <code>SELECT * FROM Numbers;</code><br><br><table border="1" style="margin-left: 10px; width: fit-content;"> <tr><td><code>digits</code></td></tr> <tr><td>-----</td></tr> <tr><td>0</td></tr> <tr><td>1</td></tr> </table> | <code>digits</code>                                                                                                                                                                                                         | -----               | 0     | 1 |   |
| <code>digits</code> |                                                                                                                                                                                                                             |                                                                                                                                                                                                                             |                     |       |   |   |
| -----               |                                                                                                                                                                                                                             |                                                                                                                                                                                                                             |                     |       |   |   |
| 0                   |                                                                                                                                                                                                                             |                                                                                                                                                                                                                             |                     |       |   |   |
| 1                   |                                                                                                                                                                                                                             |                                                                                                                                                                                                                             |                     |       |   |   |
| 4                   |                                                                                                                                                                                                                             | <code>SELECT * FROM Numbers;</code><br><br><table border="1" style="margin-left: 10px; width: fit-content;"> <tr><td><code>digits</code></td></tr> <tr><td>-----</td></tr> <tr><td>0</td></tr> <tr><td>1</td></tr> </table> | <code>digits</code> | ----- | 0 | 1 |
| <code>digits</code> |                                                                                                                                                                                                                             |                                                                                                                                                                                                                             |                     |       |   |   |
| -----               |                                                                                                                                                                                                                             |                                                                                                                                                                                                                             |                     |       |   |   |
| 0                   |                                                                                                                                                                                                                             |                                                                                                                                                                                                                             |                     |       |   |   |
| 1                   |                                                                                                                                                                                                                             |                                                                                                                                                                                                                             |                     |       |   |   |
| 5                   | <code>UPDATE Numbers SET digits=0 WHERE digits=1;</code>                                                                                                                                                                    |                                                                                                                                                                                                                             |                     |       |   |   |
| 6                   | <code>SELECT * FROM Numbers;</code><br><br><table border="1" style="margin-left: 10px; width: fit-content;"> <tr><td><code>digits</code></td></tr> <tr><td>-----</td></tr> <tr><td>0</td></tr> <tr><td>0</td></tr> </table> | <code>digits</code>                                                                                                                                                                                                         | -----               | 0     | 0 |   |
| <code>digits</code> |                                                                                                                                                                                                                             |                                                                                                                                                                                                                             |                     |       |   |   |
| -----               |                                                                                                                                                                                                                             |                                                                                                                                                                                                                             |                     |       |   |   |
| 0                   |                                                                                                                                                                                                                             |                                                                                                                                                                                                                             |                     |       |   |   |
| 0                   |                                                                                                                                                                                                                             |                                                                                                                                                                                                                             |                     |       |   |   |

| Time | User 1 action          | User 2 action                                           |
|------|------------------------|---------------------------------------------------------|
| 7    | COMMIT;                |                                                         |
| 8    |                        | Update Numbers SET digits=1 WHERE digits=0;             |
| 9    |                        | SELECT * FROM Numbers;<br><br>digits<br>-----<br>1<br>1 |
| 10   |                        | COMMIT;                                                 |
| 11   | SELECT * FROM Numbers; | <br>digits<br>-----<br>1<br>0                           |
| 12   |                        | SELECT * FROM Numbers;<br><br>digits<br>-----<br>1<br>0 |

If the same scenario is run using serializable isolation, then Amazon Redshift terminates user 2 due to a serializable violation and returns error 1023. For more information, see [How to fix serializable isolation errors \(p. 120\)](#). In this case, only user 1 can commit successfully. Not all workloads require serializable isolation as a requirement, in which case snapshot isolation suffices as the target isolation level for your database.

ARN '<ARN>'

The AWS Glue database ARN to use to create the database.

[ WITH NO DATA CATALOG SCHEMA | DATA CATALOG SCHEMA '<schema>' ]

Specifies whether to create the database using a schema from the AWS Glue Data Catalog. Must be used together with the ARN parameter.

## Syntax for using CREATE DATABASE with a datashare

The following syntax describes the CREATE DATABASE command used to create databases from a datashare for sharing data within the same AWS account.

```
CREATE DATABASE database_name
FROM DATASHARE datashare_name OF [ACCOUNT account_id] NAMESPACE namespace_guid
```

The following syntax describes the CREATE DATABASE command used to create databases from a datashare for sharing data across AWS accounts.

```
CREATE DATABASE database_name
FROM DATASHARE datashare_name OF ACCOUNT account_id NAMESPACE namespace_guid
```

## Parameters for using CREATE DATABASE with a datashare

FROM DATASHARE

A keyword that indicates where the datashare is located.

*datashare\_name*

The name of the datashare that the consumer database is created on.

NAMESPACE *namespace\_guid*

A value that specifies the producer namespace that the datashare belongs to.

ACCOUNT *account\_id*

A value that specifies the producer account that the datashare belongs to.

## Usage notes for CREATE DATABASE for data sharing

As a database superuser, when you use CREATE DATABASE to create databases from datashares within the AWS account, specify the NAMESPACE option. The ACCOUNT option is optional. When you use CREATE DATABASE to create databases from datashares across AWS accounts, specify both the ACCOUNT and NAMESPACE from the producer.

You can create only one consumer database for one datashare on a consumer cluster. You can't create multiple consumer databases referring to the same datashare.

## CREATE DATABASE limits

Amazon Redshift enforces these limits for databases:

- Maximum of 60 user-defined databases per cluster.
- Maximum of 127 bytes for a database name.
- A database name can't be a reserved word.

## Database collation

Collation is a set of rules that defines how database engine compares and sorts the character type data in SQL. Case-insensitive collation is the most commonly used collation. Amazon Redshift uses case-insensitive collation to facilitate migration from other data warehouse systems. With the native support of case-insensitive collation, Amazon Redshift continues to use important tuning or optimization methods, such as distribution keys, sort keys, or range restricted scan.

The COLLATE clause specifies the default collation for all CHAR and VARCHAR columns in the database. If CASE\_INSENSITIVE is specified, all CHAR or VARCHAR columns use case-insensitive collation. For information about collation, see [Collation sequences \(p. 601\)](#).

Data inserted or ingested in case-insensitive columns will keep its original case. But all comparison-based string operations including sorting and grouping are case-insensitive. Pattern matching operations such as LIKE predicates, similar to, and regular expression functions are also case-insensitive.

The following SQL operations support applicable collation semantics:

- Comparison operators: =, <, <=, >, >=.
- LIKE operator
- ORDER BY clauses
- GROUP BY clauses
- Aggregate functions that use string comparison, such as MIN and MAX and LISTAGG
- Window functions, such as PARTITION BY clauses and ORDER BY clauses
- Scalar functions greatest() and least(), STRPOS(), REGEXP\_COUNT(), REGEXP\_REPLACE(), REGEXP\_INSTR(), REGEXP\_SUBSTR()
- Distinct clause
- UNION, INTERSECT and EXCEPT
- IN LIST

For external queries, including Amazon Redshift Spectrum and Aurora PostgreSQL federated queries, collation of VARCHAR or CHAR column is the same as the current database-level collation.

The following example queries a Amazon Redshift Spectrum table:

```
SELECT ci_varchar FROM spectrum.test_collation
WHERE ci_varchar = 'AMAZON';

ci_varchar

amazon
Amazon
AMAZON
AmaZon
(4 rows)
```

For information on how to create tables using database collation, see [CREATE TABLE \(p. 823\)](#).

For information on the COLLATE function, see [COLLATE function \(p. 1285\)](#).

## Database collation limitations

The following are limitations when working with database collation in Amazon Redshift:

- All system tables or views, including PG catalog tables and Amazon Redshift system tables are case-sensitive.
- When consumer database and producer database have different database-level collations, Amazon Redshift doesn't support cross-database and cross-cluster queries.
- Amazon Redshift doesn't support case-insensitive collation in leader node-only query.

The following example shows an unsupported case-insensitive query and the error that Amazon Redshift sends:

```
SELECT collate(username, 'case_insensitive') FROM pg_user;
ERROR: Case insensitive collation is not supported in leader node only query.
```

- Amazon Redshift doesn't support interaction between case-sensitive and case-insensitive columns, such as comparison, function, join, or set operations.

The following examples show errors when case-sensitive and case-insensitive columns interact:

```
CREATE TABLE test
```

```
(ci_col varchar(10) COLLATE case_insensitive,
cs_col varchar(10) COLLATE case_sensitive,
cint int,
cbigint bigint);
```

```
SELECT ci_col = cs_col FROM test;
ERROR: Query with different collations is not supported yet.
```

```
SELECT concat(ci_col, cs_col) FROM test;
ERROR: Query with different collations is not supported yet.
```

```
SELECT ci_col FROM test UNION SELECT cs_col FROM test;
ERROR: Query with different collations is not supported yet.
```

```
SELECT * FROM test a, test b WHERE a.ci_col = b.cs_col;
ERROR: Query with different collations is not supported yet.
```

```
Select Coalesce(ci_col, cs_col) from test;
ERROR: Query with different collations is not supported yet.
```

```
Select case when cint > 0 then ci_col else cs_col end from test;
ERROR: Query with different collations is not supported yet.
```

- Amazon Redshift doesn't support collation for SUPER data type. Creating SUPER columns in case-insensitive databases and interactions between SUPER and case-insensitive columns aren't supported.

The following example creates a table with the SUPER as the data type in the case-insensitive database:

```
CREATE TABLE super_table (a super);
ERROR: SUPER column is not supported in case insensitive database.
```

The following example queries data with a case-insensitive string comparing with the SUPER data:

```
CREATE TABLE test_super_collation
(s super, c varchar(10) COLLATE case_insensitive, i int);
```

```
SELECT s = c FROM test_super_collation;
ERROR: Coercing from case insensitive string to SUPER is not supported.
```

To make these queries work, use the COLLATE function to convert collation of one column to match the other. For more information, see [COLLATE function \(p. 1285\)](#).

## Examples

### Creating a database

The following example creates a database named TICKIT and gives ownership to the user DWUSER.

```
create database tickit
with owner dwuser;
```

To view details about databases, query the PG\_DATABASE\_INFO catalog table.

```
select datname, datdba, datconnlimit
from pg_database_info
where datdba > 1;

datname | datdba | datconnlimit
-----+-----+-----
admin | 100 | UNLIMITED
reports | 100 | 100
ticket | 100 | 100
```

The following example creates a database named **sampledb** with SNAPSHOT isolation level.

```
CREATE DATABASE sampledb ISOLATION LEVEL SNAPSHOT;
```

The following example creates the database sales\_db from the datashare salesshare.

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF NAMESPACE
'13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

## Database collation examples

### Creating a case-insensitive database

The following example creates the sampledb database, creates the T1 table, and inserts data into the T1 table.

```
create database sampledb collate case_insensitive;
```

Connect to the new database that you just created using your SQL client. When using Amazon Redshift query editor v2, choose the sampledb in the **Editor**. When using RSQL, use a command like the following.

```
\connect sampledb;
```

```
CREATE TABLE T1 (
 col1 Varchar(20) distkey sortkey
);
```

```
INSERT INTO T1 VALUES ('bob'), ('john'), ('Mary'), ('JOHN'), ('Bob');
```

Then the query finds results with John.

```
SELECT * FROM T1 WHERE col1 = 'John';

col1

john
JOHN
(2 row)
```

### Ordering in a case-insensitive order

The following example shows the case-insensitive ordering with table T1. The ordering of *Bob* and *bob* or *John* and *john* is nondeterministic because they are equal in case-insensitive column.

```
SELECT * FROM T1 ORDER BY 1;

col1

bob
Bob
JOHN
john
Mary
(5 rows)
```

Similarly, the following example shows case-insensitive ordering with the GROUP BY clause. *Bob* and *bob* are equal and belong to the same group. It is nondeterministic which one shows up in the result.

```
SELECT col1, count(*) FROM T1 GROUP BY 1;

col1 | count
-----+-----
Mary | 1
bob | 2
JOHN | 2
(3 rows)
```

### Querying with a window function on case-insensitive columns

The following example queries a window function on a case-insensitive column.

```
SELECT col1, rank() over (ORDER BY col1) FROM T1;

col1 | rank
-----+-----
bob | 1
Bob | 1
john | 3
JOHN | 3
Mary | 5
(5 rows)
```

### Querying with the DISTINCT keyword

The following example queries the T1 table with the DISTINCT keyword.

```
SELECT DISTINCT col1 FROM T1;

col1

bob
Mary
john
(3 rows)
```

### Querying with the UNION clause

The following example shows the results from the UNION of the tables T1 and T2.

```
CREATE TABLE T2 AS SELECT * FROM T1;
```

```
SELECT col1 FROM T1 UNION SELECT col1 FROM T2;
```

```
col1

john
bob
Mary
(3 rows)
```

## CREATE DATASHARE

Creates a new datashare in the current database. The owner of this datashare is the issuer of the CREATE DATASHARE command.

Amazon Redshift associates each datashare with a single Amazon Redshift database. You can only add objects from the associated database to a datashare. You can create multiple datashares on the same Amazon Redshift database.

For information about datashares, see [Getting started data sharing \(p. 333\)](#).

To view information about the datashares, use [SHOW DATASHARES \(p. 986\)](#).

### Required privileges

Following are required privileges for CREATE DATASHARE:

- Superuser
- Users with the CREATE DATASHARE privilege
- Database owner

### Syntax

```
CREATE DATASHARE datashare_name
[[SET] PUBLICACCESSIBLE [=] TRUE | FALSE];
```

### Parameters

*datashare\_name*

The name of the datashare. The datashare name must be unique in the cluster namespace.  
[[SET] PUBLICACCESSIBLE]

A clause that specifies whether the datashare can be shared to clusters that are publicly accessible.

The default value for SET PUBLICACCESSIBLE is FALSE.

### Usage notes

By default, the owner of the datashare only owns the share but not objects within the share.

Only superusers and the database owner can use CREATE DATASHARE and delegate ALTER privileges to other users or groups.

### Examples

The following example creates the datashare salesshare.

```
CREATE DATASHARE salesshare;
```

The following example creates the datashare demoshare that AWS Data Exchange manages.

```
CREATE DATASHARE demoshare SET PUBLICACCESSIBLE TRUE, MANAGEDBY ADX;
```

## CREATE EXTERNAL FUNCTION

Creates a scalar user-defined function (UDF) based on AWS Lambda for Amazon Redshift. For more information about Lambda user-defined functions, see [Creating a scalar Lambda UDF \(p. 157\)](#).

### Required privileges

Following are required privileges for CREATE EXTERNAL FUNCTION:

- Superuser
- Users with the CREATE [ OR REPLACE ] EXTERNAL FUNCTION privilege

## Syntax

```
CREATE [OR REPLACE] EXTERNAL FUNCTION external_fn_name ([data_type] [, ...])
RETURNS data_type
VOLATILE
LAMBDA 'lambda_fn_name'
IAM_ROLE { default | 'arn:aws:iam::<AWS account-id>:role/<role-name>' }
RETRY_TIMEOUT milliseconds
MAX_BATCH_ROWS count
MAX_BATCH_SIZE size [KB | MB];
```

The following is the syntax for machine learning on Amazon Redshift. For information about the model-specific parameters, see [Parameters \(p. 891\)](#).

```
CREATE [OR REPLACE] EXTERNAL FUNCTION external_fn_name ([data_type] [, ...])
RETURNS data_type
VOLATILE
SAGEMAKER'endpoint_name'
IAM_ROLE { default | 'arn:aws:iam::<AWS account-id>:role/<role-name>' };
```

## Parameters

### OR REPLACE

A clause that specifies that if a function with the same name and input argument data types, or *signature*, as this one already exists, the existing function is replaced. You can only replace a function with a new function that defines an identical set of data types. You must be a superuser to replace a function.

If you define a function with the same name as an existing function but a different signature, you create a new function. In other words, the function name is overloaded. For more information, see [Overloading function names \(p. 161\)](#).

*external\_fn\_name*

The name of the external function. If you specify a schema name (such as myschema.myfunction), the function is created using the specified schema. Otherwise, the function is created in the current schema. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

We recommend that you prefix all UDF names with f\_. Amazon Redshift reserves the f\_ prefix for UDF names. By using the f\_ prefix, you help ensure that your UDF name won't conflict with any built-in SQL function names for Amazon Redshift now or in the future. For more information, see [Naming UDFs \(p. 161\)](#).

*data\_type*

The data type for the input arguments. For more information, see [Data types \(p. 574\)](#).

*RETURNS* *data\_type*

The data type of the value returned by the function. The RETURNS data type can be any standard Amazon Redshift data type. For more information, see [Python UDF data types \(p. 152\)](#).

*VOLATILE*

A clause that informs the query optimizer about the volatility of the function.

Given the same arguments, the function can return different results on successive calls, even for the rows in a single statement. The query optimizer can't make any assumptions about the behavior of a volatile function. A query that uses a volatile function must reevaluate the function for every input row.

The STABLE and IMMUTABLE clauses are not currently supported for Lambda UDFs.

*LAMBDA* '*lambda\_fn\_name*'

The name of the function that Amazon Redshift calls.

For steps to create an AWS Lambda function, see [Create a Lambda function with the console](#) in the [AWS Lambda Developer Guide](#).

For information regarding permissions required for the Lambda function, see [AWS Lambda permissions](#) in the [AWS Lambda Developer Guide](#).

*IAM\_ROLE* { default | 'arn:aws:iam::<AWS account-id>:role/<role-name>'}

Use the default keyword to have Amazon Redshift use the IAM role that is set as default and associated with the cluster when the CREATE EXTERNAL FUNCTION command runs.

Use the Amazon Resource Name (ARN) for an IAM role that your cluster uses for authentication and authorization. The CREATE EXTERNAL FUNCTION command is authorized to invoke Lambda functions through this IAM role. If your cluster has an existing IAM role with permissions to invoke Lambda functions attached, you can substitute your role's ARN. For more information, see [Configuring the authorization parameter for Lambda UDFs \(p. 158\)](#).

The following shows the syntax for the IAM\_ROLE parameter.

```
IAM_ROLE 'arn:aws:iam::aws-account-id:role/role-name'
```

*RETRY\_TIMEOUT* *milliseconds*

The amount of total time in milliseconds that Amazon Redshift uses for the delays in retry backoffs.

Instead of retrying immediately for any failed queries, Amazon Redshift performs backoffs and waits for a certain amount of time between retries. Then Amazon Redshift retries the request to rerun the failed query until the sum of all the delays is equal to or exceeds the RETRY\_TIMEOUT value that you specified. The default value is 20,000 milliseconds.

When a Lambda function is invoked, Amazon Redshift retries for queries that receive errors such as TooManyRequestsException, EC2ThrottledException, and ServiceException.

You can set the RETRY\_TIMEOUT parameter to 0 milliseconds to prevent any retries for a Lambda UDF.

#### **MAX\_BATCH\_ROWS** *count*

The maximum number of rows that Amazon Redshift sends in a single batch request for a single lambda invocation.

This parameter's minimum value is 1. The maximum value is INT\_MAX, or 2,147,483,647.

This parameter is optional. The default value is INT\_MAX, or 2,147,483,647.

#### **MAX\_BATCH\_SIZE** *size* [ KB | MB ]

The maximum size of the data payload that Amazon Redshift sends in a single batch request for a single lambda invocation.

This parameter's minimum value is 1 KB. The maximum value is 5 MB.

This parameter's default value is 5 MB.

KB and MB are optional. If you don't set the unit of measurement, Amazon Redshift defaults to using KB.

## Examples

Following are examples of using scalar Lambda user-defined functions (UDFs).

### Scalar Lambda UDF example using a Node.js Lambda function

The following example creates an external function called exfunc\_sum that takes two integers as input arguments. This function returns the sum as an integer output. The name of the Lambda function to be called is lambda\_sum. The language used for this Lambda function is Node.js 12.x. Make sure to specify the IAM role. The example uses 'arn:aws:iam::123456789012:user/johndoe' as the IAM role.

```
CREATE EXTERNAL FUNCTION exfunc_sum(INT,INT)
RETURNS INT
VOLATILE
LAMBDA 'lambda_sum'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test';
```

The Lambda function takes in the request payload and iterates over each row. All the values in a single row are added to calculate the sum for that row, which is saved in the response array. The number of rows in the results array is similar to the number of rows received in the request payload.

The JSON response payload must have the result data in the 'results' field for it to be recognized by the external function. The arguments field in the request sent to the Lambda function contains the data payload. There can be multiple rows in the data payload in case of a batch request. The following Lambda function iterates over all the rows in the request data payload. It also individually iterates over all the values within a single row.

```
exports.handler = async (event) => {
 // The 'arguments' field in the request sent to the Lambda function contains the data payload.
 var t1 = event['arguments'];

 // 'len(t1)' represents the number of rows in the request payload.
 // The number of results in the response payload should be the same as the number of rows received.
 const resp = new Array(t1.length);

 // Iterating over all the rows in the request payload.
```

```

for (const [i, x] of t1.entries())
{
 var sum = 0;
 // Iterating over all the values in a single row.
 for (const y of x) {
 sum = sum + y;
 }
 resp[i] = sum;
}
// The 'results' field should contain the results of the lambda call.
const response = {
 results: resp
};
return JSON.stringify(response);
};

```

The following example calls the external function with literal values.

```

select exfunc_sum(1,2);
exfunc_sum

3
(1 row)

```

The following example creates a table called `t_sum` with two columns, `c1` and `c2`, of the integer data type and inserts two rows of data. Then the external function is called by passing the column names of this table. The two table rows are sent in a batch request in request payload as a single Lambda invocation.

```

CREATE TABLE t_sum(c1 int, c2 int);
INSERT INTO t_sum VALUES (4,5), (6,7);
SELECT exfunc_sum(c1,c2) FROM t_sum;
exfunc_sum

9
13
(2 rows)

```

## Scalar Lambda UDF example using the RETRY\_TIMEOUT attribute

In the following section, you can find an example of how to use the `RETRY_TIMEOUT` attribute in Lambda UDFs.

AWS Lambda functions have concurrency limits that you can set for each function. For more information on concurrency limits, see [Managing concurrency for a Lambda function](#) in the *AWS Lambda Developer Guide* and the post [Managing AWS Lambda Function Concurrency](#) on the AWS Compute Blog.

When the number of requests being served by a Lambda UDF exceeds the concurrency limits, the new requests receive the `TooManyRequestsException` error. The Lambda UDF retries on this error until the sum of all the delays between the requests sent to the Lambda function is equal to or exceeds the `RETRY_TIMEOUT` value that you set. The default `RETRY_TIMEOUT` value is 20,000 milliseconds.

The following example uses a Lambda function named `exfunc_sleep_3`. This function takes in the request payload, iterates over each row, and converts the input to uppercase. It then sleeps for 3 seconds and returns the result. The language used for this Lambda function is Python 3.8.

The number of rows in the results array is similar to the number of rows received in the request payload. The JSON response payload must have the result data in the `results` field for it to be recognized by the external function. The `arguments` field in the request sent to the Lambda function contains the data payload. In the case of a batch request, multiple rows can appear in the data payload.

The concurrency limit for this function is specifically set to 1 in reserved concurrency to demonstrate the use of the RETRY\_TIMEOUT attribute. When the attribute is set to 1, the Lambda function can only serve one request at a time.

```
import json
import time
def lambda_handler(event, context):
 t1 = event['arguments']
 # 'len(t1)' represents the number of rows in the request payload.
 # The number of results in the response payload should be the same as the number of
 # rows received.
 resp = [None]*len(t1)

 # Iterating over all rows in the request payload.
 for i, x in enumerate(t1):
 # Iterating over all the values in a single row.
 for j, y in enumerate(x):
 resp[i] = y.upper()

 time.sleep(3)
 ret = dict()
 ret['results'] = resp
 ret_json = json.dumps(ret)
 return ret_json
```

Following, two additional examples illustrate the RETRY\_TIMEOUT attribute. They each invoke a single Lambda UDF. While invoking the Lambda UDF, each example runs the same SQL query to invoke the Lambda UDF from two concurrent database sessions at the same time. When first query that invokes the Lambda UDF is being served by the UDF, the second query receives the `TooManyRequestsException` error. This result occurs because you specifically set the reserved concurrency in the UDF to 1. For information on how to set reserved concurrency for Lambda functions, see [Configuring reserved concurrency](#).

The first example, following, sets the RETRY\_TIMEOUT attribute for the Lambda UDF to 0 milliseconds. If the Lambda request receives any exceptions from the Lambda function, Amazon Redshift doesn't make any retries. This result occurs because the RETRY\_TIMEOUT attribute is set to 0.

```
CREATE OR REPLACE EXTERNAL FUNCTION exfunc_upper(varchar)
RETURNS varchar
VOLATILE
LAMBDA 'exfunc_sleep_3'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test'
RETRY_TIMEOUT 0;
```

With the RETRY\_TIMEOUT set to 0, you can run the following two queries from separate database sessions to see different results.

The first SQL query that uses the Lambda UDF runs successfully.

```
select exfunc_upper('Varchar');
exfunc_upper

VARCHAR
(1 row)
```

The second query, which is run from a separate database session at the same time, receives the `TooManyRequestsException` error.

```
select exfunc_upper('Varchar');
```

```

ERROR: Rate Exceeded.; Exception: TooManyRequestsException; ShouldRetry: 1
DETAIL:

error: Rate Exceeded.; Exception: TooManyRequestsException; ShouldRetry: 1
code: 32103
context:query: 0
location: exfunc_client.cpp:102
process: padbmaster [pid=26384]

```

The second example, following, sets the RETRY\_TIMEOUT attribute for the Lambda UDF to 3,000 milliseconds. Even if the second query is run concurrently, the Lambda UDF retries until the total delay is 3,000 milliseconds. Thus, both queries run successfully.

```

CREATE OR REPLACE EXTERNAL FUNCTION exfunc_upper(varchar)
RETURNS varchar
VOLATILE
LAMBDA 'exfunc_sleep_3'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test'
RETRY_TIMEOUT 3000;
```

With the RETRY\_TIMEOUT set to 3,000 milliseconds, you can run the following two queries from separate database sessions to see the same results.

The first SQL query that runs the Lambda UDF runs successfully.

```

select exfunc_upper('Varchar');
exfunc_upper

VARCHAR
(1 row)
```

The second query runs concurrently, and the Lambda UDF retries until the total delay is 3,000 milliseconds.

```

select exfunc_upper('Varchar');
exfunc_upper

VARCHAR
(1 row)
```

## Scalar Lambda UDF example using a Python Lambda function

The following example creates an external function that is named `exfunc_multiplication` and that multiplies numbers and returns an integer. This example incorporates the `success` and `error_msg` fields in the Lambda response. The `success` field is set to false when there is an integer overflow in the multiplication result, and the `error_msg` message is set to `Integer multiplication overflow`. The `exfunc_multiplication` function takes three integers as input arguments and returns the sum as an integer output.

The name of the Lambda function that is called is `lambda_multiplication`. The language used for this Lambda function is Python 3.8. Make sure to specify the IAM role.

```

CREATE EXTERNAL FUNCTION exfunc_multiplication(int, int, int)
RETURNS INT
VOLATILE
LAMBDA 'lambda_multiplication'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test';
```

The Lambda function takes in the request payload and iterates over each row. All the values in a single row are multiplied to calculate the result for that row, which is saved in the response list. This example uses a Boolean success value that is set to true by default. If the multiplication result for a row has an integer overflow, then the success value is set to false. Then the iteration loop breaks.

While creating the response payload, if the success value is false, the following Lambda function adds the error\_msg field in the payload. It also sets the error message to Integer multiplication overflow. If the success value is true, then the result data is added in the results field. The number of rows in the results array, if any, is similar to the number of rows received in the request payload.

The arguments field in the request sent to the Lambda function contains the data payload. There can be multiple rows in the data payload in case of a batch request. The following Lambda function iterates over all the rows in the request data payload and individually iterates over all the values within a single row.

```
import json
def lambda_handler(event, context):
 t1 = event['arguments']
 # 'len(t1)' represents the number of rows in the request payload.
 # The number of results in the response payload should be the same as the number of
 # rows received.
 resp = [None]*len(t1)

 # By default success is set to 'True'.
 success = True
 # Iterating over all rows in the request payload.
 for i, x in enumerate(t1):
 mul = 1
 # Iterating over all the values in a single row.
 for j, y in enumerate(x):
 mul = mul*y

 # Check integer overflow.
 if (mul >= 9223372036854775807 or mul <= -9223372036854775808):
 success = False
 break
 else:
 resp[i] = mul
 ret = dict()
 ret['success'] = success
 if not success:
 ret['error_msg'] = "Integer multiplication overflow"
 else:
 ret['results'] = resp
 ret_json = json.dumps(ret)

 return ret_json
```

The following example calls the external function with literal values.

```
SELECT exfunc_multiplication(8, 9, 2);
exfunc_multiplication

144
(1 row)
```

The following example creates a table named t\_multi with three columns, c1, c2, and c3, of the integer data type. The external function is called by passing the column names of this table. The data is inserted in such a way to cause integer overflow to show how the error is propagated.

```
CREATE TABLE t_multi (c1 int, c2 int, c3 int);
```

```

INSERT INTO t_multi VALUES (2147483647, 2147483647, 4);
SELECT exfunc_multiplication(c1, c2, c3) FROM t_multi;
DETAIL:

error: Integer multiplication overflow
code: 32004context:
context:
query: 38
location: exfunc_data.cpp:276
process: query2_16_38 [pid=30494]

```

## CREATE EXTERNAL SCHEMA

Creates a new external schema in the current database. You can use this external schema to connect to Amazon RDS for PostgreSQL or Amazon Aurora PostgreSQL-Compatible Edition databases. You can also create an external schema that references a database in an external data catalog such as AWS Glue, Athena, or a database in an Apache Hive metastore, such as Amazon EMR.

The owner of this schema is the issuer of the CREATE EXTERNAL SCHEMA command. To transfer ownership of an external schema, use [ALTER SCHEMA \(p. 638\)](#) to change the owner. To grant access to the schema to other users or user groups, use the [GRANT \(p. 887\)](#) command.

You can't use the GRANT or REVOKE commands for permissions on an external table. Instead, grant or revoke the permissions on the external schema.

**Note**

If you currently have Redshift Spectrum external tables in the Amazon Athena data catalog, you can migrate your Athena data catalog to an AWS Glue Data Catalog. To use the AWS Glue Data Catalog with Redshift Spectrum, you might need to change your AWS Identity and Access Management (IAM) policies. For more information, see [Upgrading to the AWS Glue Data Catalog in the Athena User Guide](#).

To view details for external schemas, query the [SVV\\_EXTERNAL\\_SCHEMAS \(p. 1430\)](#) system view.

## Syntax

The following syntax describes the CREATE EXTERNAL SCHEMA command used to reference data using an external data catalog. For more information, see [Querying external data using Amazon Redshift Spectrum \(p. 248\)](#).

```

CREATE EXTERNAL SCHEMA [IF NOT EXISTS] local_schema_name
FROM { [DATA CATALOG] | HIVE METASTORE | POSTGRES | MYSQL | KINESIS | MSK | REDSHIFT }
[DATABASE 'database_name']
[SCHEMA 'schema_name']
[REGION 'aws-region']
[URI 'hive_metastore_uri' [PORT port_number]]
[IAM_ROLE { default | 'SESSION' | 'arn:aws:iam::<AWS account-id>:role/<role-name>' }]
[SECRET_ARN 'ssm-secret-arn']
[AUTHENTICATION { none | iam }]
[CLUSTER_ARN 'arn:aws:kafka:<region>:<AWS account-id>:cluster/msk/<cluster uuid>']
[CATALOG_ROLE { 'SESSION' | 'catalog-role-arn-string' }]
[CREATE EXTERNAL DATABASE IF NOT EXISTS]
[CATALOG_ID 'Amazon Web Services account ID containing Glue or Lake Formation database']
```

The following syntax describes the CREATE EXTERNAL SCHEMA command used to reference data using a federated query to RDS POSTGRES or Aurora PostgreSQL. You can also create an external schema that references streaming sources, such as Kinesis Data Streams. For more information, see [Querying data with federated queries in Amazon Redshift \(p. 234\)](#).

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] local_schema_name
FROM POSTGRES
DATABASE 'federated_database_name' [SCHEMA 'schema_name']
URI 'hostname' [PORT port_number]
IAM_ROLE { default | 'arn:aws:iam::<AWS account-id>:role/<role-name>' }
SECRET_ARN 'ssm-secret-arn'
```

The following syntax describes the CREATE EXTERNAL SCHEMA command used to reference data using a federated query to RDS MySQL or Aurora MySQL. For more information, see [Querying data with federated queries in Amazon Redshift \(p. 234\)](#).

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] local_schema_name
FROM MYSQL
DATABASE 'federated_database_name'
URI 'hostname' [PORT port_number]
IAM_ROLE { default | 'arn:aws:iam::<AWS account-id>:role/<role-name>' }
SECRET_ARN 'ssm-secret-arn'
```

The following syntax describes the CREATE EXTERNAL SCHEMA command used to reference data in a Kinesis stream. For more information, see [Streaming ingestion \(p. 208\)](#).

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] schema_name
FROM KINESIS
IAM_ROLE { default | 'arn:aws:iam::<AWS account-id>:role/<role-name>' }
```

The following syntax describes the CREATE EXTERNAL SCHEMA command used to reference the Amazon Managed Streaming for Apache Kafka cluster and its topics to ingest from. CLUSTER\_ARN specifies the Amazon MSK cluster that you're reading data from. For more information, see [Streaming ingestion \(p. 208\)](#).

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] schema_name
FROM MSK
IAM_ROLE { default | 'arn:aws:iam::<AWS account-id>:role/<role-name>' }
AUTHENTICATION { none | iam }
CLUSTER_ARN 'msk-cluster-arn';
```

The following syntax describes the CREATE EXTERNAL SCHEMA command used to reference data using a cross-database query.

```
CREATE EXTERNAL SCHEMA local_schema_name
FROM REDSHIFT
DATABASE 'redshift_database_name' SCHEMA 'redshift_schema_name'
```

## Parameters

### IF NOT EXISTS

A clause that indicates that if the specified schema already exists, the command should make no changes and return a message that the schema exists, rather than terminating with an error. This clause is useful when scripting, so the script doesn't fail if CREATE EXTERNAL SCHEMA tries to create a schema that already exists.

*local\_schema\_name*

The name of the new external schema. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

FROM [ DATA CATALOG ] | HIVE METASTORE | POSTGRES | MYSQL | KINESIS | MSK | REDSHIFT

A keyword that indicates where the external database is located.

DATA CATALOG indicates that the external database is defined in the Athena data catalog or the AWS Glue Data Catalog.

If the external database is defined in an external Data Catalog in a different AWS Region, the REGION parameter is required. DATA CATALOG is the default.

HIVE METASTORE indicates that the external database is defined in an Apache Hive metastore. If HIVE METASTORE is specified, URI is required.

POSTGRES indicates that the external database is defined in RDS PostgreSQL or Aurora PostgreSQL.

MYSQL indicates that the external database is defined in RDS MySQL or Aurora MySQL.

KINESIS indicates that the data source is a stream from Kinesis Data Streams.

MSK indicates that the data source is a topic from Amazon MSK.

FROM REDSHIFT

A keyword that indicates that the database is located in Amazon Redshift.

DATABASE '*redshift\_database\_name*' SCHEMA '*redshift\_schema\_name*'

The name of the Amazon Redshift database.

The *redshift\_schema\_name* indicates the schema in Amazon Redshift. The default *redshift\_schema\_name* is public.

DATABASE '*federated\_database\_name*'

A keyword that indicates the name of the external database in a supported PostgreSQL or MySQL database engine.

[SCHEMA '*schema\_name*']

The *schema\_name* indicates the schema in a supported PostgreSQL database engine. The default *schema\_name* is public.

You can't specify a SCHEMA when you set up a federated query to a supported MySQL database engine.

REGION '*aws-region*'

If the external database is defined in an Athena data catalog or the AWS Glue Data Catalog, the AWS Region in which the database is located. This parameter is required if the database is defined in an external Data Catalog.

URI '*hive\_metastore\_uri*' [ PORT *port\_number* ]

The hostname URI and port\_number of a supported PostgreSQL or MySQL database engine. The *hostname* is the head node of the replica set. The endpoint must be reachable (routable) from the Amazon Redshift cluster. The default port\_number is 5432.

If the database is in a Hive metastore, specify the URI and optionally the port number for the metastore. The default port number is 9083.

A URI doesn't contain a protocol specification ("http://"). An example valid URI: uri '172.10.10.10'.

#### Note

The supported PostgreSQL or MySQL database engine must be in the same VPC as your Amazon Redshift cluster. Create a security group linking Amazon Redshift and RDS PostgreSQL or Aurora PostgreSQL.

`IAM_ROLE { default | 'SESSION' | 'arn:aws:iam::<AWS account-id>:role/<role-name>' }`

Use the `default` keyword to have Amazon Redshift use the IAM role that is set as default and associated with the cluster when the `CREATE EXTERNAL SCHEMA` command runs.

Use `'SESSION'` if you connect to your Amazon Redshift cluster using a federated identity and access the tables from the external schema created using this command. For more information, see [Using a federated identity to manage Amazon Redshift access to local resources and Amazon Redshift Spectrum external tables](#), which explains how to configure federated identity. Note that this configuration, using `'SESSION'` in place of the ARN, can be used only if the schema is created using DATA CATALOG.

Use the Amazon Resource Name (ARN) for an IAM role that your cluster uses for authentication and authorization. As a minimum, the IAM role must have permission to perform a `LIST` operation on the Amazon S3 bucket to be accessed and a `GET` operation on the Amazon S3 objects the bucket contains. If the external database is defined in an Amazon Athena data catalog or AWS Glue Data Catalog, the IAM role must have permission to access Athena unless `CATALOG_ROLE` is specified. For more information, see [IAM policies for Amazon Redshift Spectrum \(p. 259\)](#).

The following shows the syntax for the `IAM_ROLE` parameter string for a single ARN.

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

You can chain roles so that your cluster can assume another IAM role, possibly belonging to another account. You can chain up to 10 roles. For more information, see [Chaining IAM roles in Amazon Redshift Spectrum \(p. 263\)](#).

To this IAM role, attach an IAM permissions policy similar to the following.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "AccessSecret",
 "Effect": "Allow",
 "Action": [
 "secretsmanager:GetResourcePolicy",
 "secretsmanager:GetSecretValue",
 "secretsmanager:DescribeSecret",
 "secretsmanager>ListSecretVersionIds"
],
 "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-rds-secret-VNenFy"
 },
 {
 "Sid": "VisualEditor1",
 "Effect": "Allow",
 "Action": [
 "secretsmanager:GetRandomPassword",
 "secretsmanager>ListSecrets"
],
 "Resource": "*"
 }
]
}
```

For the steps to create an IAM role to use with federated query, see [Creating a secret and an IAM role to use federated queries \(p. 238\)](#).

#### Note

Don't include spaces in the list of chained roles.

The following shows the syntax for chaining three roles.

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-1-name>,arn:aws:iam::<aws-account-id>:role/<role-2-name>,arn:aws:iam::<aws-account-id>:role/<role-3-name>'
```

SECRET\_ARN 'ssm-secret-arn'

The Amazon Resource Name (ARN) of a supported PostgreSQL or MySQL database engine secret created using AWS Secrets Manager. For information about how to create and retrieve an ARN for a secret, see [Creating a Basic Secret](#) and [Retrieving the Secret Value Secret](#) in the *AWS Secrets Manager User Guide*.

CATALOG\_ROLE { 'SESSION' | catalog-role-arn-string }

Use 'SESSION' to connect to your Amazon Redshift cluster using a federated identity for authentication and authorization to the data catalog. For more information about completing the steps for federated identity, see [Using a federated identity to manage Amazon Redshift access to local resources and Amazon Redshift Spectrum external tables](#). Note that the 'SESSION' role can be used only if the schema is created in DATA CATALOG.

Use the Amazon Resource Name ARN for an IAM role that your cluster uses for authentication and authorization for the data catalog.

If CATALOG\_ROLE isn't specified, Amazon Redshift uses the specified IAM\_ROLE. The catalog role must have permission to access the Data Catalog in AWS Glue or Athena. For more information, see [IAM policies for Amazon Redshift Spectrum \(p. 259\)](#).

The following shows the syntax for the CATALOG\_ROLE parameter string for a single ARN.

```
CATALOG_ROLE 'arn:aws:iam::<aws-account-id>:role/<catalog-role>'
```

You can chain roles so that your cluster can assume another IAM role, possibly belonging to another account. You can chain up to 10 roles. For more information, see [Chaining IAM roles in Amazon Redshift Spectrum \(p. 263\)](#).

**Note**

The list of chained roles must not include spaces.

The following shows the syntax for chaining three roles.

```
CATALOG_ROLE 'arn:aws:iam::<aws-account-id>:role/<catalog-role-1-name>,arn:aws:iam::<aws-account-id>:role/<catalog-role-2-name>,arn:aws:iam::<aws-account-id>:role/<catalog-role-3-name>'
```

## CREATE EXTERNAL DATABASE IF NOT EXISTS

A clause that creates an external database with the name specified by the DATABASE argument, if the specified external database doesn't exist. If the specified external database exists, the command makes no changes. In this case, the command returns a message that the external database exists, rather than terminating with an error.

**Note**

You can't use CREATE EXTERNAL DATABASE IF NOT EXISTS with HIVE METASTORE.

To use CREATE EXTERNAL DATABASE IF NOT EXISTS with a Data Catalog enabled for AWS Lake Formation, you need CREATE\_DATABASE permission on the Data Catalog.

CATALOG\_ID 'Amazon Web Services account ID containing Glue or Lake Formation database'

The account id where the data catalog database is stored.

CATALOG\_ID can be specified only if you plan to connect to your Amazon Redshift cluster or to Amazon Redshift Serverless using a federated identity for authentication and authorization to the data catalog by setting either of the following:

- CATALOG\_ROLE to 'SESSION'
- IAM\_ROLE to 'SESSION' and 'CATALOG\_ROLE' set to its default

For more information about completing the steps for federated identity, see [Using a federated identity to manage Amazon Redshift access to local resources and Amazon Redshift Spectrum external tables](#).

#### AUTHENTICATION

The authentication type defined for streaming ingestion. Streaming ingestion with authentication types works with Amazon Managed Streaming for Apache Kafka. The AUTHENTICATION types are the following:

- **none** – Specifies that there is no authentication step.
- **iam** – Specifies IAM authentication. When you choose this, make sure that the IAM role has permissions for IAM authentication. For more information about defining the external schema, see [Getting started with streaming ingestion from Amazon Managed Streaming for Apache Kafka \(p. 213\)](#).

#### CLUSTER\_ARN

For streaming ingestion, the cluster identifier for the Amazon Managed Streaming for Apache Kafka cluster you're streaming from. For more information, see [Streaming ingestion](#).

## Usage notes

For limits when using the Athena data catalog, see [Athena Limits](#) in the AWS General Reference.

For limits when using the AWS Glue Data Catalog, see [AWS Glue Limits](#) in the AWS General Reference.

These limits don't apply to a Hive metastore.

To unregister the schema, use the [DROP SCHEMA \(p. 872\)](#) command.

To view details for external schemas, query the following system views:

- [SVV\\_EXTERNAL\\_SCHEMAS \(p. 1430\)](#)
- [SVV\\_EXTERNAL\\_TABLES \(p. 1431\)](#)
- [SVV\\_EXTERNAL\\_COLUMNS \(p. 1428\)](#)

## Examples

The following example creates an external schema using a database in an Athena data catalog named sampledb in the US West (Oregon) Region.

```
create external schema spectrum_schema
from data catalog
database 'sampledb'
region 'us-west-2'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole';
```

The following example creates an external schema and creates a new external database named spectrum\_db.

```
create external schema spectrum_schema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
create external database if not exists;
```

The following example creates an external schema using a Hive metastore database named `hive_db`.

```
create external schema hive_schema
from hive metastore
database 'hive_db'
uri '172.10.10.10' port 99
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole';
```

The following example chains roles to use the role `myS3Role` for accessing Amazon S3 and uses `myAthenaRole` for data catalog access. For more information, see [Chaining IAM roles in Amazon Redshift Spectrum \(p. 263\)](#).

```
create external schema spectrum_schema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myRedshiftRole,arn:aws:iam::123456789012:role/
myS3Role'
catalog_role 'arn:aws:iam::123456789012:role/myAthenaRole'
create external database if not exists;
```

The following example creates an external schema that references an Aurora PostgreSQL database.

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] myRedshiftSchema
FROM POSTGRES
DATABASE 'my_aurora_db' SCHEMA 'my_aurora_schema'
URI 'endpoint to aurora hostname' PORT 5432
IAM_ROLE 'arn:aws:iam::123456789012:role/MyAuroraRole'
SECRET_ARN 'arn:aws:secretsmanager:us-east-2:123456789012:secret:development/
MyTestDatabase-AbCdEf'
```

The following example creates an external schema to refer to the `sales_db` imported on the consumer cluster.

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA 'public';
```

The following example creates an external schema that references an Aurora MySQL database.

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] myRedshiftSchema
FROM MYSQL
DATABASE 'my_aurora_db'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/MyAuroraRole'
SECRET_ARN 'arn:aws:secretsmanager:us-east-2:123456789012:secret:development/
MyTestDatabase-AbCdEf'
```

## CREATE EXTERNAL TABLE

Creates a new external table in the specified schema. All external tables must be created in an external schema. Search path isn't supported for external schemas and external tables. For more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#).

In addition to external tables created using the CREATE EXTERNAL TABLE command, Amazon Redshift can reference external tables defined in an AWS Glue or AWS Lake Formation catalog or an Apache Hive metastore. Use the [CREATE EXTERNAL SCHEMA \(p. 762\)](#) command to register an external database defined in the external catalog and make the external tables available for use in Amazon Redshift. If the external table exists in an AWS Glue or AWS Lake Formation catalog or Hive metastore, you don't need to create the table using CREATE EXTERNAL TABLE. To view external tables, query the [SVV\\_EXTERNAL\\_TABLES \(p. 1431\)](#) system view.

By running the CREATE EXTERNAL TABLE AS command, you can create an external table based on the column definition from a query and write the results of that query into Amazon S3. The results are in Apache Parquet or delimited text format. If the external table has a partition key or keys, Amazon Redshift partitions new files according to those partition keys and registers new partitions into the external catalog automatically. For more information about CREATE EXTERNAL TABLE AS, see [Usage notes \(p. 778\)](#).

You can query an external table using the same SELECT syntax you use with other Amazon Redshift tables. You can also use the INSERT syntax to write new files into the location of external table on Amazon S3. For more information, see [INSERT \(external table\) \(p. 906\)](#).

To create a view with an external table, include the WITH NO SCHEMA BINDING clause in the [CREATE VIEW \(p. 851\)](#) statement.

You can't run CREATE EXTERNAL TABLE inside a transaction (BEGIN ... END). For more information about transactions, see [Serializable isolation \(p. 119\)](#).

## Required privileges

To create external tables, you must be the owner of the external schema or a superuser. To transfer ownership of an external schema, use ALTER SCHEMA to change the owner. Access to external tables is controlled by access to the external schema. You can't [GRANT \(p. 887\)](#) or [REVOKE \(p. 916\)](#) permissions on an external table. Instead, grant or revoke USAGE on the external schema.

The [Usage notes \(p. 778\)](#) have additional information about specific permissions for external tables.

## Syntax

```
CREATE EXTERNAL TABLE
 external_schema.table_name
 (column_name data_type [, ...])
 [PARTITIONED BY (col_name data_type [, ...])]
 [{ ROW FORMAT DELIMITED row_format |
 ROW FORMAT SERDE 'serde_name' |
 [WITH SERDEPROPERTIES ('property_name' = 'property_value' [, ...])] }]
 STORED AS file_format
 LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file' }
 [TABLE PROPERTIES ('property_name'='property_value' [, ...])]
```

The following is the syntax for CREATE EXTERNAL TABLE AS.

```
CREATE EXTERNAL TABLE
 external_schema.table_name
 [PARTITIONED BY (col_name [, ...])]
 [ROW FORMAT DELIMITED row_format]
 STORED AS file_format
 LOCATION { 's3://bucket/folder/' }
 [TABLE PROPERTIES ('property_name'='property_value' [, ...])]
 AS
 { select_statement }
```

## Parameters

*external\_schema.table\_name*

The name of the table to be created, qualified by an external schema name. External tables must be created in an external schema. For more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#).

The maximum length for the table name is 127 bytes; longer names are truncated to 127 bytes. You can use UTF-8 multibyte characters up to a maximum of four bytes. Amazon Redshift enforces a limit of 9,900 tables per cluster, including user-defined temporary tables and temporary tables created by Amazon Redshift during query processing or system maintenance. Optionally, you can qualify the table name with the database name. In the following example, the database name is `spectrum_db`, the external schema name is `spectrum_schema`, and the table name is `test`.

```
create external table spectrum_db.spectrum_schema.test (c1 int)
stored as parquet
location 's3://mybucket/myfolder/';
```

If the database or schema specified doesn't exist, the table isn't created, and the statement returns an error. You can't create tables or views in the system databases `template0`, `template1`, and `padb_harvest`.

The table name must be a unique name for the specified schema.

For more information about valid names, see [Names and identifiers \(p. 571\)](#).

*(column\_name data\_type)*

The name and data type of each column being created.

The maximum length for the column name is 127 bytes; longer names are truncated to 127 bytes. You can use UTF-8 multibyte characters up to a maximum of four bytes. You can't specify column names `$path` or `$size`. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

By default, Amazon Redshift creates external tables with the pseudocolumns `$path` and `$size`. You can disable creation of pseudocolumns for a session by setting the `spectrum_enable_pseudo_columns` configuration parameter to `false`. For more information, see [Pseudocolumns \(p. 779\)](#).

If pseudocolumns are enabled, the maximum number of columns you can define in a single table is 1,598. If pseudocolumns aren't enabled, the maximum number of columns you can define in a single table is 1,600.

If you are creating a "wide table," make sure that your list of columns doesn't exceed row-width boundaries for intermediate results during loads and query processing. For more information, see [Usage notes \(p. 831\)](#).

For a CREATE EXTERNAL TABLE AS command, a column list is not required, because columns are derived from the query.

*data\_type*

The following [Data types \(p. 574\)](#) are supported:

- SMALLINT (INT2)
- INTEGER (INT, INT4)

- BIGINT (INT8)
- DECIMAL (NUMERIC)
- REAL (FLOAT4)
- DOUBLE PRECISION (FLOAT8)
- BOOLEAN (BOOL)
- CHAR (CHARACTER)
- VARCHAR (CHARACTER VARYING)
- VARBYTE (CHARACTER VARYING) – can be used with Parquet and ORC data files, and only with non-partition columns.
- DATE – can be used only with text, Parquet, or ORC data files, or as a partition column.
- TIMESTAMP

For DATE, you can use the formats as described following. For month values represented using digits, the following formats are supported:

- mm-dd-yyyy For example, 05-01-2017. This is the default.
- yyyy-mm-dd, where the year is represented by more than 2 digits. For example, 2017-05-01.

For month values represented using the three letter abbreviation, the following formats are supported:

- mmm-dd-yyyy For example, may-01-2017. This is the default.
- dd-mmm-yyyy, where the year is represented by more than 2 digits. For example, 01-may-2017.
- yyyy-mmm-dd, where the year is represented by more than 2 digits. For example, 2017-may-01.

For year values that are consistently less than 100, the year is calculated in the following manner:

- If year is less than 70, the year is calculated as the year plus 2000. For example, the date 05-01-17 in the mm-dd-yyyy format is converted into 05-01-2017.
- If year is less than 100 and greater than 69, the year is calculated as the year plus 1900. For example the date 05-01-89 in the mm-dd-yyyy format is converted into 05-01-1989.
- For year values represented by two digits, add leading zeroes to represent the year in 4 digits.

Timestamp values in text files must be in the format yyyy-mm-dd HH:mm:ss . SSSSSS, as the following timestamp value shows: 2017-05-01 11:30:59.000000.

The length of a VARCHAR column is defined in bytes, not characters. For example, a VARCHAR(12) column can contain 12 single-byte characters or 6 two-byte characters. When you query an external table, results are truncated to fit the defined column size without returning an error. For more information, see [Storage and ranges \(p. 582\)](#).

For best performance, we recommend specifying the smallest column size that fits your data. To find the maximum size in bytes for values in a column, use the [OCTET\\_LENGTH \(p. 1297\)](#) function. The following example returns the maximum size of values in the email column.

```
select max(octet_length(email)) from users;
max

62
```

PARTITIONED BY (*col\_name data\_type [, ... ]*)

A clause that defines a partitioned table with one or more partition columns. A separate data directory is used for each specified combination, which can improve query performance in some

circumstances. Partitioned columns don't exist within the table data itself. If you use a value for *col\_name* that is the same as a table column, you get an error.

After creating a partitioned table, alter the table using an [ALTER TABLE \(p. 640\)](#) ... ADD PARTITION statement to register new partitions to the external catalog. When you add a partition, you define the location of the subfolder on Amazon S3 that contains the partition data.

For example, if the table spectrum.lineitem\_part is defined with PARTITIONED BY (l\_shipdate date), run the following ALTER TABLE command to add a partition.

```
ALTER TABLE spectrum.lineitem_part ADD PARTITION (l_shipdate='1992-01-29')
LOCATION 's3://spectrum-public/lineitem_partition/l_shipdate=1992-01-29';
```

If you are using CREATE EXTERNAL TABLE AS, you don't need to run ALTER TABLE...ADD PARTITION. Amazon Redshift automatically registers new partitions in the external catalog. Amazon Redshift also automatically writes corresponding data to partitions in Amazon S3 based on the partition key or keys defined in the table.

To view partitions, query the [SVV\\_EXTERNAL\\_PARTITIONS \(p. 1429\)](#) system view.

**Note**

For a CREATE EXTERNAL TABLE AS command, you don't need to specify the data type of the partition column because this column is derived from the query.

**ROW FORMAT DELIMITED *rowformat***

A clause that specifies the format of the underlying data. Possible values for *rowformat* are as follows:

- LINES TERMINATED BY '*delimiter*'
- FIELDS TERMINATED BY '*delimiter*'

Specify a single ASCII character for '*delimiter*'. You can specify non-printing ASCII characters using octal, in the format '\ddd' where d is an octal digit (0–7) up to '\177'. The following example specifies the BEL (bell) character using octal.

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\007'
```

If ROW FORMAT is omitted, the default format is DELIMITED FIELDS TERMINATED BY '\A' (start of heading) and LINES TERMINATED BY '\n' (newline).

**ROW FORMAT SERDE '*serde\_name*' , [WITH SERDEPROPERTIES ( '*property\_name*' = '*property\_value*' [, ...] )]**

A clause that specifies the SERDE format for the underlying data.

*'serde\_name'*

The name of the SerDe. You can specify the following formats:

- org.apache.hadoop.hive.serde2.RegexSerDe
- com.amazonaws.glue.serde.GrokSerDe
- org.apache.hadoop.hive.serde2.OpenCSVSerde

This parameter supports the following SerDe property for OpenCSVSerde:

```
'wholeFile' = 'true'
```

Set the *wholeFile* property to true to properly parse new line characters (\n) within quoted strings for OpenCSV requests.

- org.openx.data.jsonserde.JsonSerDe
  - The JSON SERDE also supports Ion files.
  - The JSON must be well-formed.
  - Timestamps in Ion and JSON must use ISO8601 format.
  - This parameter supports the following SerDe property for JsonSerDe:

```
'strip.outer.array'='true'
```

Processes Ion/JSON files containing one very large array enclosed in outer brackets ([ ... ]) as if it contains multiple JSON records within the array.

- com.amazon.ionhiveserde.IonHiveSerDe

The Amazon ION format provides text and binary formats, in addition to data types. For an external table that references data in ION format, you map each column in the external table to the corresponding element in the ION format data. For more information, see [Amazon Ion](#). You also need to specify the input and output formats.

WITH SERDEPROPERTIES ('*property\_name*' = '*property\_value*' [, ...] ) ]

Optionally, specify property names and values, separated by commas.

If ROW FORMAT is omitted, the default format is DELIMITED FIELDS TERMINATED BY '\A' (start of heading) and LINES TERMINATED BY '\n' (newline).

STORED AS *file\_format*

The file format for data files.

Valid formats are as follows:

- PARQUET
- RCFILE (for data using ColumnarSerDe only, not LazyBinaryColumnarSerDe)
- SEQUENCEFILE
- TEXTFILE
- ORC
- AVRO
- INPUTFORMAT '*input\_format\_classname*' OUTPUTFORMAT '*output\_format\_classname*'

The CREATE EXTERNAL TABLE AS command only supports two file formats, TEXTFILE and PARQUET.

For INPUTFORMAT and OUTPUTFORMAT, specify a class name, as the following example shows.

```
'org.apache.hadoop.mapred.TextInputFormat'
```

LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest\_file'}

The path to the Amazon S3 bucket or folder that contains the data files or a manifest file that contains a list of Amazon S3 object paths. The buckets must be in the same AWS Region as the Amazon Redshift cluster. For a list of supported AWS Regions, see [Amazon Redshift Spectrum considerations \(p. 249\)](#).

If the path specifies a bucket or folder, for example 's3://mybucket/custdata/', Redshift Spectrum scans the files in the specified bucket or folder and any subfolders. Redshift Spectrum ignores hidden files and files that begin with a period or underscore.

If the path specifies a manifest file, the 's3://bucket/manifest\_file' argument must explicitly reference a single file—for example, 's3://mybucket/manifest.txt'. It can't reference a key prefix.

The manifest is a text file in JSON format that lists the URL of each file that is to be loaded from Amazon S3 and the size of the file, in bytes. The URL includes the bucket name and full object path for the file. The files that are specified in the manifest can be in different buckets, but all the buckets must be in the same AWS Region as the Amazon Redshift cluster. If a file is listed twice, the file is loaded twice. The following example shows the JSON for a manifest that loads three files.

```
{
 "entries": [
 {"url": "s3://mybucket-alpha/custdata.1", "meta": { "content_length": 5956875 } },
 {"url": "s3://mybucket-alpha/custdata.2", "meta": { "content_length": 5997091 } },
 {"url": "s3://mybucket-beta/custdata.1", "meta": { "content_length": 5978675 } }
]
}
```

You can make the inclusion of a particular file mandatory. To do this, include a `mandatory` option at the file level in the manifest. When you query an external table with a mandatory file that is missing, the `SELECT` statement fails. Ensure that all files included in the definition of the external table are present. If they aren't all present, an error appears showing the first mandatory file that isn't found. The following example shows the JSON for a manifest with the `mandatory` option set to `true`.

```
{
 "entries": [
 {"url": "s3://mybucket-alpha/custdata.1", "mandatory": true, "meta": {
 "content_length": 5956875 } },
 {"url": "s3://mybucket-alpha/custdata.2", "mandatory": false, "meta": {
 "content_length": 5997091 } },
 {"url": "s3://mybucket-beta/custdata.1", "meta": { "content_length": 5978675 } }
]
}
```

To reference files created using `UNLOAD`, you can use the manifest created using [UNLOAD \(p. 991\)](#) with the `MANIFEST` parameter. The manifest file is compatible with a manifest file for [COPY from Amazon S3 \(p. 683\)](#), but uses different keys. Keys that aren't used are ignored.

`TABLE PROPERTIES ( 'property_name'='property_value' [, ...] )`

A clause that sets the table definition for table properties.

**Note**

Table properties are case-sensitive.

`'compression_type'='value'`

A property that sets the type of compression to use if the file name doesn't contain an extension. If you set this property and there is a file extension, the extension is ignored and the value set by the property is used. Valid values for compression type are as follows:

- bzip2
- gzip
- none
- snappy

`'data_cleansing_enabled'='true / false'`

This property sets whether data handling is on for the table. When `'data_cleansing_enabled'` is set to `true`, data handling is on for the table. When `'data_cleansing_enabled'` is set to `false`, data handling is off for the table. Following is a list of the table-level data handling properties controlled by this property:

- `column_count_mismatch_handling`
- `invalid_char_handling`

- numeric\_overflow\_handling
- replacement\_char
- surplus\_char\_handling

For examples, see [Data handling examples \(p. 786\)](#).

'invalid\_char\_handling'='value'

Specifies the action to perform when query results contain invalid UTF-8 character values. You can specify the following actions:

DISABLED

Doesn't perform invalid character handling.

FAIL

Cancels queries that return data containing invalid UTF-8 values.

SET\_TO\_NULL

Replaces invalid UTF-8 values with null.

DROP\_ROW

Replaces each value in the row with null.

REPLACE

Replaces the invalid character with the replacement character you specify using replacement\_char.

'replacement\_char'='character'

Specifies the replacement character to use when you set invalid\_char\_handling to REPLACE.

'numeric\_overflow\_handling'='value'

Specifies the action to perform when ORC data contains an integer (for example, BIGINT or int64) that is larger than the column definition (for example, SMALLINT or int16). You can specify the following actions:

DISABLED

Invalid character handling is turned off.

FAIL

Cancel the query when the data includes invalid characters.

SET\_TO\_NULL

Set invalid characters to null.

DROP\_ROW

Set each value in the row to null.

'surplus\_bytes\_handling'='value'

Specifies how to handle data being loaded that exceeds the length of the data type defined for columns containing VARBYTE data. By default, Redshift Spectrum sets the value to null for data that exceeds the width of the column.

You can specify the following actions to perform when the query returns data that exceeds the length of the data type:

**SET\_TO\_NULL**

Replaces data that exceeds the column width with null.

**DISABLED**

Doesn't perform surplus byte handling.

**FAIL**

Cancels queries that return data exceeding the column width.

**DROP\_ROW**

Drop all rows that contain data exceeding column width.

**TRUNCATE**

Removes the characters that exceed the maximum number of characters defined for the column.

**'surplus\_char\_handling'='value'**

Specifies how to handle data being loaded that exceeds the length of the data type defined for columns containing VARCHAR, CHAR, or string data. By default, Redshift Spectrum sets the value to null for data that exceeds the width of the column.

You can specify the following actions to perform when the query returns data that exceeds the column width:

**SET\_TO\_NULL**

Replaces data that exceeds the column width with null.

**DISABLED**

Doesn't perform surplus character handling.

**FAIL**

Cancels queries that return data exceeding the column width.

**DROP\_ROW**

Replaces each value in the row with null.

**TRUNCATE**

Removes the characters that exceed the maximum number of characters defined for the column.

**'column\_count\_mismatch\_handling'='value'**

Identifies if the file contains less or more values for a row than the number of columns specified in the external table definition. This property is only available for an uncompressed text file format. You can specify the following actions:

**DISABLED**

Column count mismatch handling is turned off.

**FAIL**

Fail the query if the column count mismatch is detected.

**SET\_TO\_NULL**

Fill missing values with NULL and ignore the additional values in each row.

**DROP\_ROW**

Drop all rows that contain column count mismatch error from the scan.

`'numRows'='row_count'`

A property that sets the numRows value for the table definition. To explicitly update an external table's statistics, set the numRows property to indicate the size of the table. Amazon Redshift doesn't analyze external tables to generate the table statistics that the query optimizer uses to generate a query plan. If table statistics aren't set for an external table, Amazon Redshift generates a query execution plan based on an assumption that external tables are the larger tables and local tables are the smaller tables.

`'skip.header.line.count'='line_count'`

A property that sets number of rows to skip at the beginning of each source file.

`'serialization.null.format'=''`

A property that specifies Spectrum should return a NULL value when there is an exact match with the text supplied in a field.

`'orc.schema.resolution'='mapping_type'`

A property that sets the column mapping type for tables that use ORC data format. This property is ignored for other data formats.

Valid values for column mapping type are as follows:

- name
- position

If the `orc.schema.resolution` property is omitted, columns are mapped by name by default. If `orc.schema.resolution` is set to any value other than '`name`' or '`position`', columns are mapped by position. For more information about column mapping, see [Mapping external table columns to ORC columns \(p. 282\)](#).

**Note**

The COPY command maps to ORC data files only by position. The `orc.schema.resolution` table property has no effect on COPY command behavior.

`'write.parallel'='on / off'`

A property that sets whether CREATE EXTERNAL TABLE AS should write data in parallel. By default, CREATE EXTERNAL TABLE AS writes data in parallel to multiple files, according to the number of slices in the cluster. The default option is on. When '`write.parallel`' is set to off, CREATE EXTERNAL TABLE AS writes to one or more data files serially onto Amazon S3. This table property also applies to any subsequent INSERT statement into the same external table.

`'write.maxfilesize.mb'='size'`

A property that sets the maximum size (in MB) of each file written to Amazon S3 by CREATE EXTERNAL TABLE AS. The size must be a valid integer between 5 and 6200. The default maximum file size is 6,200 MB. This table property also applies to any subsequent INSERT statement into the same external table.

`'write.kms.key.id'='value'`

You can specify an AWS Key Management Service key to enable Server-Side Encryption (SSE) for Amazon S3 objects, where `value` is one of the following:

- `auto` to use the default AWS KMS key stored in the Amazon S3 bucket.
- `kms-key` that you specify to encrypt data.

`select_statement`

A statement that inserts one or more rows into the external table by defining any query. All rows that the query produces are written to Amazon S3 in either text or Parquet format based on the table definition.

## Examples

A collection of examples is available at [Examples \(p. 780\)](#).

## Usage notes

This topic contains usage notes for [CREATE EXTERNAL TABLE \(p. 768\)](#). You can't view details for Amazon Redshift Spectrum tables using the same resources that you use for standard Amazon Redshift tables, such as [PG\\_TABLE\\_DEF \(p. 1677\)](#), [STV\\_TBL\\_PERM \(p. 1599\)](#), PG\_CLASS, or information\_schema. If your business intelligence or analytics tool doesn't recognize Redshift Spectrum external tables, configure your application to query [SVV\\_EXTERNAL\\_TABLES \(p. 1431\)](#) and [SVV\\_EXTERNAL\\_COLUMNS \(p. 1428\)](#).

## CREATE EXTERNAL TABLE AS

In some cases, you might run the CREATE EXTERNAL TABLE AS command on an AWS Glue Data Catalog, AWS Lake Formation external catalog, or Apache Hive metastore. In such cases, you use an AWS Identity and Access Management (IAM) role to create the external schema. This IAM role must have both read and write permissions on Amazon S3.

If you use a Lake Formation catalog, the IAM role must have the permission to create table in the catalog. In this case, it must also have the data lake location permission on the target Amazon S3 path. This IAM role becomes the owner of the new AWS Lake Formation table.

To ensure that file names are unique, Amazon Redshift uses the following format for the name of each file uploaded to Amazon S3 by default.

`<date>_<time>_<microseconds>_<query_id>_<slice-number>_part_<part-number>.<format>.`

An example is `20200303_004509_810669_1007_0001_part_00.parquet`.

Consider the following when running the CREATE EXTERNAL TABLE AS command:

- The Amazon S3 location must be empty.
- Amazon Redshift only supports PARQUET and TEXTFILE formats when using the STORED AS clause.
- You don't need to define a column definition list. Column names and column data types of the new external table are derived directly from the SELECT query.
- You don't need to define the data type of the partition column in the PARTITIONED BY clause. If you specify a partition key, the name of this column must exist in the SELECT query result. When having multiple partition columns, their order in the SELECT query doesn't matter. Amazon Redshift uses their order defined in the PARTITIONED BY clause to create the external table.
- Amazon Redshift automatically partitions output files into partition folders based on the partition key values. By default, Amazon Redshift removes partition columns from the output files.
- The LINES TERMINATED BY 'delimiter' clause isn't supported.
- The ROW FORMAT SERDE 'serde\_name' clause isn't supported.
- The use of manifest files isn't supported. Thus, you can't define the LOCATION clause to a manifest file on Amazon S3.
- Amazon Redshift automatically updates the 'numRows' table property at the end of the command.
- The 'compression\_type' table property only accepts 'none' or 'snappy' for the PARQUET file format.
- Amazon Redshift doesn't allow the LIMIT clause in the outer SELECT query. Instead, you can use a nested LIMIT clause.
- You can use STL\_UNLOAD\_LOG to track the files that are written to Amazon S3 by each CREATE EXTERNAL TABLE AS operation.

## Permissions to create and query external tables

To create external tables, make sure that you're the owner of the external schema or a superuser. To transfer ownership of an external schema, use [ALTER SCHEMA \(p. 638\)](#). The following example changes the owner of the spectrum\_schema schema to newowner.

```
alter schema spectrum_schema owner to newowner;
```

To run a Redshift Spectrum query, you need the following permissions:

- Usage permission on the schema
- Permission to create temporary tables in the current database

The following example grants usage permission on the schema spectrum\_schema to the spectrumusers user group.

```
grant usage on schema spectrum_schema to group spectrumusers;
```

The following example grants temporary permission on the database spectrumdb to the spectrumusers user group.

```
grant temp on database spectrumdb to group spectrumusers;
```

## Pseudocolumns

By default, Amazon Redshift creates external tables with the pseudocolumns \$path and \$size. Select these columns to view the path to the data files on Amazon S3 and the size of the data files for each row returned by a query. The \$path and \$size column names must be delimited with double quotation marks. A *SELECT \** clause doesn't return the pseudocolumns. You must explicitly include the \$path and \$size column names in your query, as the following example shows.

```
select "$path", "$size"
from spectrum.sales_part
where saledate = '2008-12-01';
```

You can disable creation of pseudocolumns for a session by setting the *spectrum\_enable\_pseudo\_columns* configuration parameter to *false*.

### Important

Selecting \$size or \$path incurs charges because Redshift Spectrum scans the data files in Amazon S3 to determine the size of the result set. For more information, see [Amazon Redshift Pricing](#).

## Setting data handling options

You can set table parameters to specify input handling for data being queried in external tables, including:

- Surplus characters in columns containing VARCHAR, CHAR, and string data. For more information, see the external table property *surplus\_char\_handling*.
- Invalid characters in columns containing VARCHAR, CHAR, and string data. For more information, see the external table property *invalid\_char\_handling*.
- Replacement character to use when you specify REPLACE for the external table property *invalid\_char\_handling*.

- Cast overflow handling in columns containing integer and decimal data. For more information, see the external table property `numeric_overflow_handling`.
- `Surplus_bytes_handling` to specify input handling for surplus bytes in columns containing varbyte data. For more information, see the external table property `surplus_bytes_handling`.

## Examples

The following example creates a table named SALES in the Amazon Redshift external schema named spectrum. The data is in tab-delimited text files. The TABLE PROPERTIES clause sets the numRows property to 170,000 rows.

Depending on the identity you use to run CREATE EXTERNAL TABLE, there may be IAM permissions that you have to configure. As a best practice, we recommend attaching permissions policies to an IAM role and then assigning it to users and groups as needed. For more information, see [Identity and access management in Amazon Redshift](#).

```
create external table spectrum.sales(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
saledate date,
qtypaid smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://awssampledbuswest2/ticket/spectrum/sales/'
table properties ('numRows'='170000');
```

The following example creates a table that uses the JsonSerDe to reference data in JSON format.

```
create external table spectrum.cloudtrail_json (
event_version int,
event_id bigint,
event_time timestamp,
event_type varchar(10),
awsregion varchar(20),
event_name varchar(max),
event_source varchar(max),
requesttime timestamp,
useragent varchar(max),
recipientaccountid bigint)
row format serde 'org.openx.data.jsonserde.JsonSerDe'
with serdeproperties (
'dots.in.keys' = 'true',
'mapping.requesttime' = 'requesttimestamp'
) location 's3://mybucket/json/cloudtrail';
```

The following CREATE EXTERNAL TABLE AS example creates a nonpartitioned external table. Then it writes the result of the SELECT query as Apache Parquet to the target Amazon S3 location.

```
CREATE EXTERNAL TABLE spectrum.lineitem
STORED AS parquet
LOCATION 'S3://mybucket/cetas/lineitem/'
AS SELECT * FROM local_lineitem;
```

The following example creates a partitioned external table and includes the partition columns in the SELECT query.

```
CREATE EXTERNAL TABLE spectrum.partitioned_lineitem
PARTITIONED BY (l_shipdate, l_shipmode)
STORED AS parquet
LOCATION 'S3://mybucket/cetas/partitioned_lineitem/'
AS SELECT l_orderkey, l_shipmode, l_shipdate, l_partkey FROM local_table;
```

For a list of existing databases in the external data catalog, query the [SVV\\_EXTERNAL\\_DATABASES \(p. 1429\)](#) system view.

```
select eskind,databasename,esoptions from svv_external_databases order by databasename;
```

| eskind         | databasename | esoptions                                                                         |
|----------------|--------------|-----------------------------------------------------------------------------------|
| 1   default    |              | {"REGION":"us-west-2","IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"} |
| 1   sampledb   |              | {"REGION":"us-west-2","IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"} |
| 1   spectrumdb |              | {"REGION":"us-west-2","IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"} |

To view details of external tables, query the [SVV\\_EXTERNAL\\_TABLES \(p. 1431\)](#) and [SVV\\_EXTERNAL\\_COLUMNS \(p. 1428\)](#) system views.

The following example queries the SVV\_EXTERNAL\_TABLES view.

```
select schemaname, tablename, location from svv_external_tables;
```

| schemaname | tablename  | location                                                |
|------------|------------|---------------------------------------------------------|
| spectrum   | sales      | s3://awssampledbuswest2/ticket/spectrum/sales           |
| spectrum   | sales_part | s3://awssampledbuswest2/ticket/spectrum/sales_partition |

The following example queries the SVV\_EXTERNAL\_COLUMNS view.

```
select * from svv_external_columns where schemaname like 'spectrum%' and tablename = 'sales';
```

| schemaname | tablename | columnname | external_type | columnnum | part_key |
|------------|-----------|------------|---------------|-----------|----------|
| spectrum   | sales     | salesid    | int           | 1         | 0        |
| spectrum   | sales     | listid     | int           | 2         | 0        |
| spectrum   | sales     | sellerid   | int           | 3         | 0        |
| spectrum   | sales     | buyerid    | int           | 4         | 0        |
| spectrum   | sales     | eventid    | int           | 5         | 0        |
| spectrum   | sales     | saledate   | date          | 6         | 0        |
| spectrum   | sales     | qtysold    | smallint      | 7         | 0        |
| spectrum   | sales     | pricepaid  | decimal(8,2)  | 8         | 0        |
| spectrum   | sales     | commission | decimal(8,2)  | 9         | 0        |
| spectrum   | sales     | saletime   | timestamp     | 10        | 0        |

To view table partitions, use the following query.

```
select schemaname, tablename, values, location
```

```
from svv_external_partitions
where tablename = 'sales_part';
```

| schemaname | tablename  | values           | location                                                                 |
|------------|------------|------------------|--------------------------------------------------------------------------|
| spectrum   | sales_part | [ "2008-01-01" ] | s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-01 |
| spectrum   | sales_part | [ "2008-02-01" ] | s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-02 |
| spectrum   | sales_part | [ "2008-03-01" ] | s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-03 |
| spectrum   | sales_part | [ "2008-04-01" ] | s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-04 |
| spectrum   | sales_part | [ "2008-05-01" ] | s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-05 |
| spectrum   | sales_part | [ "2008-06-01" ] | s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-06 |
| spectrum   | sales_part | [ "2008-07-01" ] | s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-07 |
| spectrum   | sales_part | [ "2008-08-01" ] | s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-08 |
| spectrum   | sales_part | [ "2008-09-01" ] | s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-09 |
| spectrum   | sales_part | [ "2008-10-01" ] | s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-10 |
| spectrum   | sales_part | [ "2008-11-01" ] | s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-11 |
| spectrum   | sales_part | [ "2008-12-01" ] | s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-12 |

The following example returns the total size of related data files for an external table.

```
select distinct "$path", "$size"
 from spectrum.sales_part;

$path | $size
-----+-----+
s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-01/ | 1616
s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-02/ | 1444
s3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-02/ | 1444
```

## Partitioning examples

To create an external table partitioned by date, run the following command.

```
create external table spectrum.sales_part(
 salesid integer,
 listid integer,
 sellerid integer,
 buyerid integer,
 eventid integer,
 dateid smallint,
 qtysold smallint,
 pricepaid decimal(8,2),
 commission decimal(8,2),
 saletime timestamp)
partitioned by (saledate date)
row format delimited
fields terminated by '|'
stored as textfile
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/'
```

```
table properties ('numRows'='170000');
```

To add the partitions, run the following ALTER TABLE commands.

```
alter table spectrum.sales_part
add if not exists partition (saledate='2008-01-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-01/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-02-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-02/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-03-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-03/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-04-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-04/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-05-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-05/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-06-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-06/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-07-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-07/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-08-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-08/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-09-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-09/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-10-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-10/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-11-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-11/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-12-01')
location 's3://awssampledbuswest2/ticket/spectrum/sales_partition/saledate=2008-12/';
```

To select data from the partitioned table, run the following query.

```
select top 10 spectrum.sales_part.eventid, sum(spectrum.sales_part.pricepaid)
from spectrum.sales_part, event
where spectrum.sales_part.eventid = event.eventid
 and spectrum.sales_part.pricepaid > 30
 and saledate = '2008-12-01'
group by spectrum.sales_part.eventid
order by 2 desc;
```

| eventid | sum      |
|---------|----------|
| 914     | 36173.00 |
| 5478    | 27303.00 |
| 5061    | 26383.00 |
| 4406    | 26252.00 |
| 5324    | 24015.00 |
| 1829    | 23911.00 |
| 3601    | 23616.00 |
| 3665    | 23214.00 |
| 6069    | 22869.00 |

|                 |
|-----------------|
| 5638   22551.00 |
|-----------------|

To view external table partitions, query the [SVV\\_EXTERNAL\\_PARTITIONS \(p. 1429\)](#) system view.

```
select schemaname, tablename, values, location from svv_external_partitions
where tablename = 'sales_part';
```

| schemaname      | tablename        | values           | location                                 |
|-----------------|------------------|------------------|------------------------------------------|
| -----           | -----            | -----            | -----                                    |
| spectrum        | sales_part       | [ "2008-01-01" ] | s3://awssampledbuswest2/ticket/spectrum/ |
| sales_partition | saledate=2008-01 |                  |                                          |
| spectrum        | sales_part       | [ "2008-02-01" ] | s3://awssampledbuswest2/ticket/spectrum/ |
| sales_partition | saledate=2008-02 |                  |                                          |
| spectrum        | sales_part       | [ "2008-03-01" ] | s3://awssampledbuswest2/ticket/spectrum/ |
| sales_partition | saledate=2008-03 |                  |                                          |
| spectrum        | sales_part       | [ "2008-04-01" ] | s3://awssampledbuswest2/ticket/spectrum/ |
| sales_partition | saledate=2008-04 |                  |                                          |
| spectrum        | sales_part       | [ "2008-05-01" ] | s3://awssampledbuswest2/ticket/spectrum/ |
| sales_partition | saledate=2008-05 |                  |                                          |
| spectrum        | sales_part       | [ "2008-06-01" ] | s3://awssampledbuswest2/ticket/spectrum/ |
| sales_partition | saledate=2008-06 |                  |                                          |
| spectrum        | sales_part       | [ "2008-07-01" ] | s3://awssampledbuswest2/ticket/spectrum/ |
| sales_partition | saledate=2008-07 |                  |                                          |
| spectrum        | sales_part       | [ "2008-08-01" ] | s3://awssampledbuswest2/ticket/spectrum/ |
| sales_partition | saledate=2008-08 |                  |                                          |
| spectrum        | sales_part       | [ "2008-09-01" ] | s3://awssampledbuswest2/ticket/spectrum/ |
| sales_partition | saledate=2008-09 |                  |                                          |
| spectrum        | sales_part       | [ "2008-10-01" ] | s3://awssampledbuswest2/ticket/spectrum/ |
| sales_partition | saledate=2008-10 |                  |                                          |
| spectrum        | sales_part       | [ "2008-11-01" ] | s3://awssampledbuswest2/ticket/spectrum/ |
| sales_partition | saledate=2008-11 |                  |                                          |
| spectrum        | sales_part       | [ "2008-12-01" ] | s3://awssampledbuswest2/ticket/spectrum/ |
| sales_partition | saledate=2008-12 |                  |                                          |

## Row format examples

The following shows an example of specifying the ROW FORMAT SERDE parameters for data files stored in AVRO format.

```
create external table spectrum.sales(salesid int, listid int, sellerid int, buyerid int,
eventid int, dateid int, qtysold int, pricepaid decimal(8,2), comment VARCHAR(255))
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
WITH SERDEPROPERTIES ('avro.schema.literal'='{"namespace": "dory.sample", "name": "dory_avro", "type": "record", "fields": [{"name": "salesid", "type": "int"}, {"name": "listid", "type": "int"}, {"name": "sellerid", "type": "int"}, {"name": "buyerid", "type": "int"}, {"name": "eventid", "type": "int"}, {"name": "dateid", "type": "int"}, {"name": "qtysold", "type": "int"}, {"name": "pricepaid", "type": {"type": "bytes", "logicalType": "decimal", "precision": 8, "scale": 2}}, {"name": "comment", "type": "string"}]')
STORED AS AVRO
location 's3://mybucket/avro/sales' ;
```

The following shows an example of specifying the ROW FORMAT SERDE parameters using RegEx.

```
create external table spectrum.types(
cbigint bigint,
```

```
cbigint_null bigint,
cint int,
cint_null int)
row format serde 'org.apache.hadoop.hive.serde2.RegexSerDe'
with serdeproperties ('input.regex'='([^\x01]+)\x01([^\x01]+)\x01([^\x01]+)\x01([^\x01]+)')
stored as textfile
location 's3://mybucket/regex/types';
```

The following shows an example of specifying the ROW FORMAT SERDE parameters using Grok.

```
create external table spectrum.grok_log(
timestamp varchar(255),
pid varchar(255),
loglevel varchar(255),
progname varchar(255),
message varchar(255))
row format serde 'com.amazonaws.glue.serde.GrokSerDe'
with serdeproperties ('input.format'='[DFEWI], \\\%{TIMESTAMP_ISO8601:timestamp} # %{POSINT:pid:int}\\] *(<loglevel>:DEBUG|FATAL|ERROR|WARN|INFO) -- +%{DATA:progname}: %{GREEDYDATA:message}')
stored as textfile
location 's3://mybucket/grok/logs';
```

The following shows an example of defining an Amazon S3 server access log in an S3 bucket. You can use Redshift Spectrum to query Amazon S3 access logs.

```
CREATE EXTERNAL TABLE spectrum.mybucket_s3_logs(
bucketowner varchar(255),
bucket varchar(255),
requestdatetime varchar(2000),
remoteip varchar(255),
requester varchar(255),
requested varchar(255),
operation varchar(255),
key varchar(255),
requesturi_operation varchar(255),
requesturi_key varchar(255),
requesturi_httpprotoversion varchar(255),
httpstatus varchar(255),
errorcode varchar(255),
bytesent bigint,
objectsize bigint,
totaltime varchar(255),
turnaroundtime varchar(255),
referrer varchar(255),
useragent varchar(255),
versionid varchar(255)
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
'input.regex' = '([^\n]*)([^\n]*)\\[(.*?)\\]([^]*)([^]*)([^]*)([^]*)([^]*)([^]*)([^]*)([^]*)([^]*)([^]*)\\s*([^\n]*)\\s*([^\n]*)\"(- |[^]*)([^]*)([^]*)([^]*)([^]*)([^]*)([^]*)([^]*)([^]*)([^]*)([^]*)\" ([^\n]*)([^]*)*$')
LOCATION 's3://mybucket/s3logs';
```

The following shows an example of specifying the ROW FORMAT SERDE parameters for ION format data.

```
CREATE EXTERNAL TABLE tbl_name (columns)
ROW FORMAT SERDE 'com.amazon ionhiveserde.IonHiveSerDe'
STORED AS
INPUTFORMAT 'com.amazon ionhiveserde.formats.IonInputFormat'
```

```
OUTPUTFORMAT 'com.amazon ionhiveserde.formats.IonOutputFormat'
LOCATION 's3://s3-bucket/prefix'
```

## Data handling examples

The following examples access the file: [spi\\_global\\_rankings.csv](#). You can upload the `spi_global_rankings.csv` file to an Amazon S3 bucket to try these examples.

The following example creates the external schema `schema_spectrum_uddh` and database `spectrum_db_uddh`. For `aws-account-id`, enter your AWS account ID and for `role-name` enter your Redshift Spectrum role name.

```
create external schema schema_spectrum_uddh
from data catalog
database 'spectrum_db_uddh'
iam_role 'arn:aws:iam::aws-account-id:role/role-name'
create external database if not exists;
```

The following example creates the external table `soccer_league` in the external schema `schema_spectrum_uddh`.

```
CREATE EXTERNAL TABLE schema_spectrum_uddh.soccer_league
(
 league_rank smallint,
 prev_rank smallint,
 club_name varchar(15),
 league_name varchar(20),
 league_off decimal(6,2),
 league_def decimal(6,2),
 league_spi decimal(6,2),
 league_nspl integer
)
ROW FORMAT DELIMITED
 FIELDS TERMINATED BY ','
 LINES TERMINATED BY '\n\1'
stored as textfile
LOCATION 's3://spectrum-uddh/league/'
table properties ('skip.header.line.count'='1');
```

Check the number of rows in the `soccer_league` table.

```
select count(*) from schema_spectrum_uddh.soccer_league;
```

The numbers of rows displays.

```
count
645
```

The following query displays the top 10 clubs. Because club Barcelona has an invalid character in the string, a NULL is displayed for the name.

```
select league_rank,club_name,league_name,league_nspl
from schema_spectrum_uddh.soccer_league
where league_rank between 1 and 10;
```

```
league_rank club_name league_name league_nspl
```

|    |                                      |       |
|----|--------------------------------------|-------|
| 1  | Manchester City Barclays Premier Lea | 34595 |
| 2  | Bayern Munich German Bundesliga      | 34151 |
| 3  | Liverpool Barclays Premier Lea       | 33223 |
| 4  | Chelsea Barclays Premier Lea         | 32808 |
| 5  | Ajax Dutch Eredivisie                | 32790 |
| 6  | Atletico Madrid Spanish Primera Divi | 31517 |
| 7  | Real Madrid Spanish Primera Divi     | 31469 |
| 8  | NULL Spanish Primera Divi            | 31321 |
| 9  | RB Leipzig German Bundesliga         | 31014 |
| 10 | Paris Saint-Ger French Ligue 1       | 30929 |

The following example alters the soccer\_league table to specify the invalid\_char\_handling, replacement\_char, and data\_cleansing\_enabled external table properties to insert a question mark (?) as a substitute for unexpected characters.

```
alter table schema_spectrum_uddh.soccer_league
set table properties
('invalid_char_handling'='REPLACE', 'replacement_char'='?', 'data_cleansing_enabled'='true');
```

The following example queries the table soccer\_league for teams with a rank from 1 to 10.

```
select league_rank,club_name,league_name,league_nspi
from schema_spectrum_uddh.soccer_league
where league_rank between 1 and 10;
```

Because the table properties were altered, the results show the top 10 clubs, with the question mark (?) replacement character in the eighth row for club Barcelona.

| league_rank | club_name                            | league_name | league_nspi |
|-------------|--------------------------------------|-------------|-------------|
| 1           | Manchester City Barclays Premier Lea | 34595       |             |
| 2           | Bayern Munich German Bundesliga      | 34151       |             |
| 3           | Liverpool Barclays Premier Lea       | 33223       |             |
| 4           | Chelsea Barclays Premier Lea         | 32808       |             |
| 5           | Ajax Dutch Eredivisie                | 32790       |             |
| 6           | Atletico Madrid Spanish Primera Divi | 31517       |             |
| 7           | Real Madrid Spanish Primera Divi     | 31469       |             |
| 8           | Barcel?na Spanish Primera Divi       | 31321       |             |
| 9           | RB Leipzig German Bundesliga         | 31014       |             |
| 10          | Paris Saint-Ger French Ligue 1       | 30929       |             |

The following example alters the soccer\_league table to specify the invalid\_char\_handling external table properties to drop rows with unexpected characters.

```
alter table schema_spectrum_uddh.soccer_league
set table properties ('invalid_char_handling'='DROP_ROW', 'data_cleansing_enabled'='true');
```

The following example queries the table soccer\_league for teams with a rank from 1 to 10.

```
select league_rank,club_name,league_name,league_nspi
from schema_spectrum_uddh.soccer_league
where league_rank between 1 and 10;
```

The results display the top clubs, not including the eighth row for club Barcelona.

| league_rank | club_name       | league_name          | league_nspi |
|-------------|-----------------|----------------------|-------------|
| 1           | Manchester City | Barclays Premier Lea | 34595       |

|    |                 |                      |       |
|----|-----------------|----------------------|-------|
| 2  | Bayern Munich   | German Bundesliga    | 34151 |
| 3  | Liverpool       | Barclays Premier Lea | 33223 |
| 4  | Chelsea         | Barclays Premier Lea | 32808 |
| 5  | Ajax            | Dutch Eredivisie     | 32790 |
| 6  | Atletico Madrid | Spanish Primera Divi | 31517 |
| 7  | Real Madrid     | Spanish Primera Divi | 31469 |
| 9  | RB Leipzig      | German Bundesliga    | 31014 |
| 10 | Paris Saint-Ger | French Ligue 1       | 30929 |

## CREATE FUNCTION

Creates a new scalar user-defined function (UDF) using either a SQL SELECT clause or a Python program.

### Required privileges

Following are required privileges for CREATE FUNCTION:

- For CREATE FUNCTION:
  - Superuser can use both trusted and untrusted languages to create functions.
  - Users with the CREATE [ OR REPLACE ] FUNCTION privilege can create functions with trusted languages.
- For REPLACE FUNCTION:
  - Superuser
  - Users with the CREATE [ OR REPLACE ] FUNCTION privilege
  - Function owner

## Syntax

```
CREATE [OR REPLACE] FUNCTION f_function_name
({ [py_arg_name py_arg_data_type |
 sql_arg_data_type] [, ...] })
RETURNS data_type
{ VOLATILE | STABLE | IMMUTABLE }
AS $$
{ python_program | SELECT_clause }
$$ LANGUAGE { plpythonu | sql }
```

## Parameters

### OR REPLACE

Specifies that if a function with the same name and input argument data types, or *signature*, as this one already exists, the existing function is replaced. You can only replace a function with a new function that defines an identical set of data types. You must be a superuser to replace a function.

If you define a function with the same name as an existing function but a different signature, you create a new function. In other words, the function name is overloaded. For more information, see [Overloading function names \(p. 161\)](#).

### f\_function\_name

The name of the function. If you specify a schema name (such as myschema.myfunction), the function is created using the specified schema. Otherwise, the function is created in the current schema. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

We recommend that you prefix all UDF names with f\_. Amazon Redshift reserves the f\_ prefix for UDF names, so by using the f\_ prefix, you ensure that your UDF name will not conflict with any existing or future Amazon Redshift built-in SQL function names. For more information, see [Naming UDFs \(p. 161\)](#).

You can define more than one function with the same function name if the data types for the input arguments are different. In other words, the function name is overloaded. For more information, see [Overloading function names \(p. 161\)](#).

*py\_arg\_name py\_arg\_data\_type / sql\_arg\_data\_type*

For a Python UDF, a list of input argument names and data types. For a SQL UDF, a list of data types, without argument names. In a Python UDF, refer to arguments using the argument names. In a SQL UDF, refer to arguments using \$1, \$2, and so on, based on the order of the arguments in the argument list.

For a SQL UDF, the input and return data types can be any standard Amazon Redshift data type. For a Python UDF, the input and return data types can be SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE PRECISION, BOOLEAN, CHAR, VARCHAR, DATE, or TIMESTAMP. In addition, Python user-defined functions (UDFs) support a data type of ANYELEMENT. This is automatically converted to a standard data type based on the data type of the corresponding argument supplied at runtime. If multiple arguments use ANYELEMENT, they all resolve to the same data type at runtime, based on the first ANYELEMENT argument in the list. For more information, see [Python UDF data types \(p. 152\)](#) and [Data types \(p. 574\)](#).

You can specify a maximum of 32 arguments.

**RETURNS** *data\_type*

The data type of the value returned by the function. The RETURNS data type can be any standard Amazon Redshift data type. In addition, Python UDFs can use a data type of ANYELEMENT, which is automatically converted to a standard data type based on the argument supplied at runtime. If you specify ANYELEMENT for the return data type, at least one argument must use ANYELEMENT. The actual return data type matches the data type supplied for the ANYELEMENT argument when the function is called. For more information, see [Python UDF data types \(p. 152\)](#).

**VOLATILE | STABLE | IMMUTABLE**

Informs the query optimizer about the volatility of the function.

You will get the best optimization if you label your function with the strictest volatility category that is valid for it. However, if the category is too strict, there is a risk that the optimizer will erroneously skip some calls, resulting in an incorrect result set. In order of strictness, beginning with the least strict, the volatility categories are as follows:

- VOLATILE
- STABLE
- IMMUTABLE

**VOLATILE**

Given the same arguments, the function can return different results on successive calls, even for the rows in a single statement. The query optimizer can't make any assumptions about the behavior of a volatile function, so a query that uses a volatile function must reevaluate the function for every input row.

**STABLE**

Given the same arguments, the function is guaranteed to return the same results for all rows processed within a single statement. The function can return different results when called in different statements. This category allows the optimizer to optimize multiple calls of the function within a single statement to a single call for the statement.

## IMMUTABLE

Given the same arguments, the function always returns the same result, forever. When a query calls an IMMUTABLE function with constant arguments, the optimizer pre-evaluates the function.

AS \$\$ *statement* \$\$

A construct that encloses the statement to be run. The literal keywords AS \$\$ and \$\$ are required.

Amazon Redshift requires you to enclose the statement in your function by using a format called dollar quoting. Anything within the enclosure is passed exactly as is. You don't need to escape any special characters because the contents of the string are written literally.

With *dollar quoting*, you use a pair of dollar signs (\$\$) to signify the start and the end of the statement to run, as shown in the following example.

```
$$ my statement $$
```

Optionally, between the dollar signs in each pair, you can specify a string to help identify the statement. The string that you use must be the same in both the start and the end of the enclosure pairs. This string is case-sensitive, and it follows the same constraints as an unquoted identifier except that it can't contain dollar signs. The following example uses the string test.

```
$test$ my statement $test$
```

For more information about dollar quoting, see "Dollar-quoted String Constants" under [Lexical Structure](#) in the PostgreSQL documentation.

*python\_program*

A valid executable Python program that returns a value. The statement that you pass in with the function must conform to indentation requirements as specified in the [Style Guide for Python Code](#) on the Python website. For more information, see [Python language support for UDFs \(p. 153\)](#).

*SQL\_clause*

A SQL SELECT clause.

The SELECT clause can't include any of the following types of clauses:

- FROM
- INTO
- WHERE
- GROUP BY
- ORDER BY
- LIMIT

*LANGUAGE { plpythonu | sql }*

For Python, specify plpythonu. For SQL, specify sql. You must have permission for usage on language for SQL or plpythonu. For more information, see [UDF security and privileges \(p. 150\)](#).

## Usage notes

### Nested functions

You can call another SQL user-defined function (UDF) from within a SQL UDF. The nested function must exist when you run the CREATE FUNCTION command. Amazon Redshift doesn't track dependencies for UDFs, so if you drop the nested function, Amazon Redshift doesn't return an error. However, the UDF will

fail if the nested function doesn't exist. For example, the following function calls the `f_sql_greater` function in the SELECT clause.

```
create function f_sql_commission (float, float)
 returns float
stable
as $$
 select f_sql_greater ($1, $2)
$$ language sql;
```

## UDF security and privileges

To create a UDF, you must have permission for usage on language for SQL or plpythonu (Python). By default, USAGE ON LANGUAGE SQL is granted to PUBLIC. However, you must explicitly grant USAGE ON LANGUAGE PLPYTHONU to specific users or groups.

To revoke usage for SQL, first revoke usage from PUBLIC. Then grant usage on SQL only to the specific users or groups permitted to create SQL UDFs. The following example revokes usage on SQL from PUBLIC then grants usage to the user group `udf_devs`.

```
revoke usage on language sql from PUBLIC;
grant usage on language sql to group udf_devs;
```

To run a UDF, you must have execute permission for each function. By default, execute permission for new UDFs is granted to PUBLIC. To restrict usage, revoke execute permission from PUBLIC for the function. Then grant the privilege to specific individuals or groups.

The following example revokes execute permission on function `f_py_greater` from PUBLIC then grants usage to the user group `udf_devs`.

```
revoke execute on function f_py_greater(a float, b float) from PUBLIC;
grant execute on function f_py_greater(a float, b float) to group udf_devs;
```

Superusers have all privileges by default.

For more information, see [GRANT \(p. 887\)](#) and [REVOKE \(p. 916\)](#).

## Examples

### Scalar Python UDF example

The following example creates a Python UDF that compares two integers and returns the larger value.

```
create function f_py_greater (a float, b float)
 returns float
stable
as $$
 if a > b:
 return a
 return b
$$ language plpythonu;
```

The following example queries the SALES table and calls the new `f_py_greater` function to return either COMMISSION or 20 percent of PRICEPAID, whichever is greater.

```
select f_py_greater (commission, pricepaid*0.20) from sales;
```

## Scalar SQL UDF example

The following example creates a function that compares two numbers and returns the larger value.

```
create function f_sql_greater (float, float)
 returns float
stable
as $$
 select case when $1 > $2 then $1
 else $2
 end
$$ language sql;
```

The following query calls the new f\_sql\_greater function to query the SALES table and returns either COMMISSION or 20 percent of PRICEPAID, whichever is greater.

```
select f_sql_greater (commission, pricepaid*0.20) from sales;
```

## CREATE GROUP

Defines a new user group. Only a superuser can create a group.

### Syntax

```
CREATE GROUP group_name
[[WITH] [USER username] [, ...]]
```

### Parameters

*group\_name*

Name of the new user group. Group names beginning with two underscores are reserved for Amazon Redshift internal use. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

WITH

Optional syntax to indicate additional parameters for CREATE GROUP.

USER

Add one or more users to the group.

*username*

Name of the user to add to the group.

### Examples

The following example creates a user group named ADMIN\_GROUP with a two users, ADMIN1 and ADMIN2.

```
create group admin_group with user admin1, admin2;
```

## CREATE IDENTITY PROVIDER

Defines a new identity provider. Only a superuser can create an identity provider.

## Syntax

```
CREATE IDENTITY PROVIDER identity_provider_name TYPE type_name
NAMESPACE namespace_name
PARAMETERS parameter_string
```

## Parameters

*identity\_provider\_name*

Name of the new identity provider. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

*type\_name*

The identity provider to interface with. Azure is currently the only supported identity provider.

*namespace\_name*

The namespace. This is a unique, shorthand identifier for the identity provider directory.

*parameter\_string*

A string containing a properly formatted JSON object that contains parameters and values required for the identity provider.

## Example

The following example creates an identity provider named *oauth\_standard*, with a TYPE *azure*, to establish communication with Microsoft Azure Active Directory (AD).

```
CREATE IDENTITY PROVIDER oauth_standard TYPE azure
NAMESPACE 'aad'
PARAMETERS '{"issuer":"https://sts.windows.net/2sdfdsf-d475-420d-b5ac-667adad7c702/",
"client_id":"87f4aa26-78b7-410e-bf29-57b39929ef9a",
"client_secret":"BUAH~ewiqewrqwerUUY^%tHe1oNZShoiU7",
"audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift"]
}'
```

## CREATE LIBRARY

Installs a Python library, which is available for users to incorporate when creating a user-defined function (UDF) with the [CREATE FUNCTION \(p. 788\)](#) command. The total size of user-installed libraries can't exceed 100 MB.

CREATE LIBRARY can't be run inside a transaction block (BEGIN ... END). For more information about transactions, see [Serializable isolation \(p. 119\)](#).

Amazon Redshift supports Python version 2.7. For more information, see [www.python.org](http://www.python.org).

For more information, see [Importing custom Python library modules \(p. 154\)](#).

## Required privileges

Following are required privileges for CREATE LIBRARY:

- Superuser

- Users with the CREATE LIBRARY privilege or with the privilege of the specified language

## Syntax

```
CREATE [OR REPLACE] LIBRARY library_name LANGUAGE plpythonu
FROM
{ 'https://file_url'

| 's3://bucketname/file_name'

authorization

[REGION [AS] 'aws_region']

IAM_ROLE { default | 'arn:aws:iam::<AWS account-id>:role/<role-name>' }

}
```

## Parameters

### OR REPLACE

Specifies that if a library with the same name as this one already exists, the existing library is replaced. REPLACE commits immediately. If a UDF that depends on the library is running concurrently, the UDF might fail or return unexpected results, even if the UDF is running within a transaction. You must be the owner or a superuser to replace a library.

### *library\_name*

The name of the library to be installed. You can't create a library that contains a module with the same name as a Python Standard Library module or an Amazon Redshift preinstalled Python module. If an existing user-installed library uses the same Python package as the library to be installed, you must drop the existing library before installing the new library. For more information, see [Python language support for UDFs \(p. 153\)](#).

### LANGUAGE plpythonu

The language to use. Python (plpythonu) is the only supported language. Amazon Redshift supports Python version 2.7. For more information, see [www.python.org](http://www.python.org).

### FROM

The location of the library file. You can specify an Amazon S3 bucket and object name, or you can specify a URL to download the file from a public website. The library must be packaged in the form of a .zip file. For more information, see [Building and Installing Python Modules](#) in the Python documentation.

#### `https://file_url`

The URL to download the file from a public website. The URL can contain up to three redirects. The following is an example of a file URL.

```
'https://www.example.com/pylib.zip'
```

#### `s3://bucket_name/file_name`

The path to a single Amazon S3 object that contains the library file. The following is an example of an Amazon S3 object path.

```
's3://mybucket/my-pylib.zip'
```

If you specify an Amazon S3 bucket, you must also provide credentials for an AWS user that has permission to download the file.

**Important**

If the Amazon S3 bucket doesn't reside in the same AWS Region as your Amazon Redshift cluster, you must use the REGION option to specify the AWS Region in which the data is located. The value for *aws\_region* must match an AWS Region listed in the table in the [REGION \(p. 686\)](#) parameter description for the COPY command.

*authorization*

A clause that indicates the method your cluster uses for authentication and authorization to access the Amazon S3 bucket that contains the library file. Your cluster must have permission to access the Amazon S3 with the LIST and GET actions.

The syntax for authorization is the same as for the COPY command authorization. For more information, see [Authorization parameters \(p. 693\)](#).

```
IAM_ROLE { default | 'arn:aws:iam::<AWS account-id>:role/<role-name>' }
```

Use the default keyword to have Amazon Redshift use the IAM role that is set as default and associated with the cluster when the CREATE LIBRARY command runs.

Use the Amazon Resource Name (ARN) for an IAM role that your cluster uses for authentication and authorization. If you specify IAM\_ROLE, you can't use ACCESS\_KEY\_ID and SECRET\_ACCESS\_KEY, SESSION\_TOKEN, or CREDENTIALS.

Optionally, if the Amazon S3 bucket uses server-side encryption, provide the encryption key in the credentials-args string. If you use temporary security credentials, provide the temporary token in the credentials-args string.

For more information, see [Temporary security credentials \(p. 716\)](#).

*REGION [AS] aws\_region*

The AWS Region where the Amazon S3 bucket is located. REGION is required when the Amazon S3 bucket isn't in the same AWS Region as the Amazon Redshift cluster. The value for *aws\_region* must match an AWS Region listed in the table in the [REGION \(p. 686\)](#) parameter description for the COPY command.

By default, CREATE LIBRARY assumes that the Amazon S3 bucket is located in the same AWS Region as the Amazon Redshift cluster.

## Examples

The following two examples install the [urllibparse](#) Python module, which is packaged in a file named `urlparse3-1.0.3.zip`.

The following command installs a UDF library named `f_urllibparse` from a package that has been uploaded to an Amazon S3 bucket located in the US East Region.

```
create library f_urllibparse
language plpythonu
from 's3://mybucket/urlparse3-1.0.3.zip'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
region as 'us-east-1';
```

The following example installs a library named `f_urllibparse` from a library file on a website.

```
create library f_urllibparse
```

```
language plpythonu
from 'https://example.com/packages/urlparse3-1.0.3.zip';
```

## CREATE MASKING POLICY

Creates a new dynamic data masking policy to obfuscate data of a given format. For more information on dynamic data masking, see [Dynamic data masking \(p. 555\)](#).

Superusers and users or roles that have the sys:secadmin role can create a masking policy.

### Syntax

```
CREATE MASKING POLICY [IF NOT EXISTS]
 policy_name
 WITH (input_columns)
 USING (masking_expression);
```

### Parameters

*masking\_policy\_name*

The name of the masking policy. The masking policy can't have the same name as another masking policy that already exists in the database.

*input\_columns*

A tuple of column names in the format (col1 type, col2 type ...).

Column names are used as the input for the masking expression. Column names don't have to match the names of the columns being masked, but the input and output data types must match.

*masking\_expression*

The SQL expression used to transform the target columns. It can be written using data manipulation functions such as String manipulation functions, or in conjunction with user-defined functions written in SQL, Python, or with AWS Lambda. You can include a tuple of column expressions for masking policies that have multiple outputs. If you use a constant as your masking expression, you must explicitly cast it to a type that matches the input type.

You must have the USAGE permission on any user-defined functions that you use in the masking expression.

## CREATE MATERIALIZED VIEW

Creates a materialized view based on one or more Amazon Redshift tables. You can also base materialized views on external tables created using Spectrum or federated query. For information about Spectrum, see [Querying external data using Amazon Redshift Spectrum \(p. 248\)](#). For information about federated query, see [Querying data with federated queries in Amazon Redshift \(p. 234\)](#).

### Syntax

```
CREATE MATERIALIZED VIEW mv_name
[BACKUP { YES | NO }]
[table_attributes]
[AUTO REFRESH { YES | NO }]
```

AS query

## Parameters

### BACKUP

A clause that specifies whether the materialized view is included in automated and manual cluster snapshots, which are stored in Amazon S3.

The default value for BACKUP is YES.

You can specify BACKUP NO to save processing time when creating snapshots and restoring from snapshots, and to reduce the amount of storage required in Amazon S3.

#### Note

The BACKUP NO setting has no effect on automatic replication of data to other nodes within the cluster, so tables with BACKUP NO specified are restored in a node failure.

### *table\_attributes*

A clause that specifies how the data in the materialized view is distributed, including the following:

- The distribution style for the materialized view, in the format DISTSTYLE { EVEN | ALL | KEY }. If you omit this clause, the distribution style is EVEN. For more information, see [Distribution styles \(p. 51\)](#).
- The distribution key for the materialized view, in the format DISTKEY ( *distkey\_identifier* ). For more information, see [Designating distribution styles \(p. 53\)](#).
- The sort key for the materialized view, in the format SORTKEY ( *column\_name* [, ...] ). For more information, see [Working with sort keys \(p. 60\)](#).

### AS query

A valid SELECT statement that defines the materialized view and its content. The result set from the query defines the columns and rows of the materialized view. For information about limitations when creating materialized views, see [Limitations \(p. 798\)](#).

Furthermore, specific SQL language constructs used in the query determines whether the materialized view can be incrementally or fully refreshed. For information about the refresh method, see [REFRESH MATERIALIZED VIEW \(p. 914\)](#). For information about the limitations for incremental refresh, see [Limitations for incremental refresh \(p. 915\)](#).

If the query contains an SQL command that doesn't support incremental refresh, Amazon Redshift displays a message indicating that the materialized view will use a full refresh. The message may or may not be displayed, depending on the SQL client application. Check the state column of the [STV\\_MV\\_INFO \(p. 1588\)](#) to see the refresh type used by a materialized view.

### AUTO REFRESH

A clause that defines whether the materialized view should be automatically refreshed with latest changes from its base tables. The default value is NO. For more information, see [Refreshing a materialized view \(p. 202\)](#).

## Usage notes

To create a materialized view, you must have the following privileges:

- CREATE privileges for a schema.
- Table-level or column-level SELECT privilege on the base tables to create a materialized view. If you have column-level privileges on specific columns, you can create a materialized view on only those columns.

## DDL updates to materialized views or base tables

When using materialized views in Amazon Redshift, follow these usage notes for data definition language (DDL) updates to materialized views or base tables.

- You can add columns to a base table without affecting any materialized views that reference the base table.
- Some operations can leave the materialized view in a state that can't be refreshed at all. Examples are operations such as renaming or dropping a column, changing the type of a column, and changing the name of a schema. Such materialized views can be queried but can't be refreshed. In this case, you must drop and recreate the materialized view.
- In general, you can't alter a materialized view's definition (its SQL statement).
- You can't rename a materialized view.

## Limitations

You can't define a materialized view that references or includes any of the following:

- Standard views, or system tables and views.
- Temporary tables.
- User-defined functions.
- The ORDER BY, LIMIT, or OFFSET clause.
- Late binding references to base tables. In other words, any base tables or related columns referenced in the defining SQL query of the materialized view must exist and must be valid.
- Leader node-only functions: CURRENT\_SCHEMA, CURRENT\_SCHEMAS, HAS\_DATABASE\_PRIVILEGE, HAS\_SCHEMA\_PRIVILEGE, HAS\_TABLE\_PRIVILEGE.
- You can't use the AUTO REFRESH YES option when the materialized view definition includes mutable functions or external schemas. You also can't use it when you define a materialized view on another materialized view.
- You don't have to manually run [ANALYZE \(p. 663\)](#) on materialized views. This happens currently only via AUTO ANALYZE. For more information, see [Analyzing tables \(p. 105\)](#).

## Examples

The following example creates a materialized view from three base tables that are joined and aggregated. Each row represents a category with the number of tickets sold. When you query the tickets\_mv materialized view, you directly access the precomputed data in the tickets\_mv materialized view.

```
CREATE MATERIALIZED VIEW tickets_mv AS
 select catgroup,
 sum(qtysold) as sold
 from category c, event e, sales s
 where c.catid = e.catid
 and e.eventid = s.eventid
 group by catgroup;
```

The following example creates a materialized view similar to the previous example and uses the aggregate function MAX().

```
CREATE MATERIALIZED VIEW tickets_mv_max AS
 select catgroup,
```

```

max(qtysold) as sold
from category c, event e, sales s
where c.catid = e.catid
and e.eventid = s.eventid
group by catgroup;

SELECT name, state FROM STV_MV_INFO;

```

The following example uses a UNION ALL clause to join the Amazon Redshift public\_sales table and the Redshift Spectrum spectrum.sales table to create a material view mv\_sales\_vw. For information about the CREATE EXTERNAL TABLE command for Amazon Redshift Spectrum, see [CREATE EXTERNAL TABLE \(p. 768\)](#). The Redshift Spectrum external table references the data on Amazon S3.

```

CREATE MATERIALIZED VIEW mv_sales_vw as
select salesid, qtysold, pricepaid, commission, saletime from public.sales
union all
select salesid, qtysold, pricepaid, commission, saletime from spectrum.sales

```

The following example creates a materialized view mv\_fq based on a federated query external table. For information about federated query, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#).

```

CREATE MATERIALIZED VIEW mv_fq as select firstname, lastname from apg.mv_fq_example;

select firstname, lastname from mv_fq;
firstname | lastname
-----+-----
John | Day
Jane | Doe
(2 rows)

```

The following example shows the definition of a materialized view.

```

SELECT pg_catalog.pg_get_viewdef('mv_sales_vw'::regclass::oid, true);

pg_get_viewdef

create materialized view mv_sales_vw as select a from t;

```

The following sample shows how to set AUTO REFRESH in the materialized view definition and also specifies a DISTSTYLE. First, create a simple base table.

```
CREATE TABLE baseball_table (ball int, bat int);
```

Then, create a materialized view.

```

CREATE MATERIALIZED VIEW mv_baseball DISTSTYLE ALL AUTO REFRESH YES AS SELECT ball AS
baseball FROM baseball_table;

```

Now you can query the mv\_baseball materialized view. To check if AUTO REFRESH is turned on for a materialized view, see [STV\\_MV\\_INFO \(p. 1588\)](#).

For details about materialized view overview and SQL commands used to refresh and drop materialized views, see the following topics:

- [Creating materialized views in Amazon Redshift \(p. 198\)](#)
- [REFRESH MATERIALIZED VIEW \(p. 914\)](#)

- [DROP MATERIALIZED VIEW \(p. 869\)](#)

## CREATE MODEL

### Topics

- [Prerequisites \(p. 800\)](#)
- [Required privileges \(p. 800\)](#)
- [Cost control \(p. 800\)](#)
- [Full CREATE MODEL \(p. 801\)](#)
- [Parameters \(p. 801\)](#)
- [Usage notes \(p. 804\)](#)
- [Use cases \(p. 805\)](#)

## Prerequisites

Before you use the CREATE MODEL statement, complete the prerequisites in [Cluster setup for using Amazon Redshift ML \(p. 401\)](#). The following is a high-level summary of the prerequisites.

- Create an Amazon Redshift cluster with the AWS Management Console or the AWS Command Line Interface (AWS CLI).
- Attach the AWS Identity and Access Management (IAM) policy while creating the cluster.
- To allow Amazon Redshift and SageMaker to assume the role to interact with other services, add the appropriate trust policy to the IAM role.

For details for the IAM role, trust policy, and other prerequisites, see [Cluster setup for using Amazon Redshift ML \(p. 401\)](#).

Following, you can find different use cases for the CREATE MODEL statement.

- [Simple CREATE MODEL \(p. 805\)](#)
- [CREATE MODEL with user guidance \(p. 806\)](#)
- [CREATE XGBoost models with AUTO OFF \(p. 809\)](#)
- [Bring your own model \(BYOM\) - local inference \(p. 811\)](#)
- [CREATE MODEL with K-MEANS \(p. 814\)](#)
- [Full CREATE MODEL \(p. 801\)](#)

## Required privileges

Following are required privileges for CREATE MODEL:

- Superuser
- Users with the CREATE MODEL privilege
- Roles with the GRANT CREATE MODEL privilege

## Cost control

Amazon Redshift ML uses existing cluster resources to create prediction models, so you don't have to pay additional costs. However, you might have additional costs if you need to resize your cluster or want

to train your models. Amazon Redshift ML uses Amazon SageMaker to train models, which does have an additional associated cost. There are ways to control additional costs, such as limiting the maximum amount of time training can take or by limiting the number of training examples used to train your model. For more information, see [Costs for using Amazon Redshift ML](#).

## Full CREATE MODEL

The following summarizes the basic options of the full CREATE MODEL syntax.

### Full CREATE MODEL syntax

The following is the full syntax of the CREATE MODEL statement.

#### Important

When creating a model using the CREATE MODEL statement, follow the order of the keywords in the syntax following.

```
CREATE MODEL model_name
 FROM { table_name | (select_statement) | 'job_name' }
 [TARGET column_name]
 FUNCTION function_name (data_type [, ...])
 IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
 [AUTO ON / OFF]
 -- default is AUTO ON
 [MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER | KMEANS }]
 -- not required for non AUTO OFF case, default is the list of all supported types
 -- required for AUTO OFF
 [PROBLEM_TYPE (REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION)]
 -- not supported when AUTO OFF
 [OBJECTIVE ('MSE' | 'Accuracy' | 'F1' | 'F1_Macro' | 'AUC' |
 'reg:squarederror' | 'reg:squaredlogerror' | 'reg:logistic' |
 'reg:pseudohubererror' | 'reg:tweedie' | 'binary:logistic' |
 'binary:hinge',
 'multi:softmax')]
 -- for AUTO ON: first 5 are valid
 -- for AUTO OFF: 6-13 are valid
 [PREPROCESSORS 'string']
 -- required for AUTO OFF, when it has to be 'none'
 -- optional for AUTO ON
 [HYPERPARAMETERS { DEFAULT | DEFAULT EXCEPT (Key 'value' (,...)) }]
 -- support XGBoost hyperparameters, except OBJECTIVE
 -- required and only allowed for AUTO OFF
 -- default NUM_ROUND is 100
 -- NUM_CLASS is required if objective is multi:softmax (only possible for AUTO OFF)
 [SETTINGS (
 S3_BUCKET 'bucket', |
 -- required
 KMS_KEY_ID 'kms_string', |
 -- optional
 S3_GARBAGE_COLLECT on / off, |
 -- optional, default is on.
 MAX_CELLS integer, |
 -- optional, default is 1,000,000
 MAX_RUNTIME integer (, ...)
 -- optional, default is 5400 (1.5 hours)
)]
```

## Parameters

### model\_name

The name of the model. The model name in a schema must be unique.

`FROM { table_name | ( select_query ) | 'job_name' }`

The `table_name` or the query that specifies the training data. They can either be an existing table in the system, or an Amazon Redshift-compatible SELECT query enclosed with parentheses, that is `( )`. There must be at least two columns in the query result.

`TARGET column_name`

The name of the column that becomes the prediction target. The column must exist in the `FROM` clause.

`FUNCTION function_name ( data_type [, ...] )`

The name of the function to be created and the data types of the input arguments. You can provide a schema name.

`IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }`

Use the `default` keyword to have Amazon Redshift use the IAM role that is set as default and associated with the cluster when the `CREATE MODEL` command runs. Alternatively, you can specify an ARN of an IAM role to use that role.

`[ AUTO ON / OFF ]`

Turns on or off `CREATE MODEL` automatic discovery of preprocessor, algorithm, and hyper-parameters selection.

`MODEL_TYPE { XGBOOST / MLP / LINEAR_LEARNER / KMEANS }`

(Optional) Specifies the model type. You can specify if you want to train a model of a specific model type, such as XGBoost, multilayer perceptron (MLP), KMEANS, or Linear Learner, which are all algorithms that Amazon SageMaker Autopilot supports. If you don't specify the parameter, then all supported model types are searched during training for the best model.

`PROBLEM_TYPE ( REGRESSION / BINARY_CLASSIFICATION / MULTICLASS_CLASSIFICATION )`

(Optional) Specifies the problem type. If you know the problem type, you can restrict Amazon Redshift to only search of the best model of that specific model type. If you don't specify this parameter, a problem type is discovered during the training, based on your data.

`OBJECTIVE ( 'MSE' | 'Accuracy' | 'F1' | 'F1Macro' | 'AUC' | 'reg:squarederror' | 'reg:squaredlogerror' | 'reg:logistic' | 'reg:pseudohubererror' | 'reg:tweedie' | 'binary:logistic' | 'binary:hinge' | 'multi:softmax' | 'rank:pairwise' | 'rank:ndcg' )`

(Optional) Specifies the name of the objective metric used to measure the predictive quality of a machine learning system. This metric is optimized during training to provide the best estimate for model parameter values from data. If you don't specify a metric explicitly, the default behavior is to automatically use MSE: for regression, F1: for binary classification, Accuracy: for multiclass classification. For more information about objectives, see [AutoMLJobObjective](#) in the [Amazon SageMaker API Reference](#) and [Learning task parameters](#) in the XGBOOST documentation.

`PREPROCESSORS 'string'`

(Optional) Specifies certain combinations of preprocessors to certain sets of columns. The format is a list of `columnSets`, and the appropriate transforms to be applied to each set of columns. Amazon Redshift applies all the transformers in a specific `transformers` list to all columns in the corresponding `ColumnSet`. For example, to apply `OneHotEncoder` with `Imputer` to columns `t1` and `t2`, use the sample command following.

```
CREATE MODEL customer_churn
FROM customer_data
TARGET 'Churn'
FUNCTION predict_churn
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
```

```

PROBLEM_TYPE BINARY_CLASSIFICATION
OBJECTIVE 'F1'
PREPROCESSORS [
...
 {"ColumnSet": [
 "t1",
 "t2"
],
 "Transformers": [
 "OneHotEncoder",
 "Imputer"
]
},
 {"ColumnSet": [
 "t3"
],
 "Transformers": [
 "OneHotEncoder"
]
},
 {"ColumnSet": [
 "temp"
],
 "Transformers": [
 "Imputer",
 "NumericPassthrough"
]
}
]
SETTINGS (
 S3_BUCKET 'bucket'
)

```

#### HYPERPARAMETERS { DEFAULT | DEFAULT EXCEPT ( key 'value' (...) ) }

Specifies whether the default XGBoost parameters are used or overridden by user-specified values. The values must be enclosed with single quotes. Following are examples of parameters for XGBoost and their defaults.

| Parameter name   | Parameter value | Default value                                       | Notes                        |
|------------------|-----------------|-----------------------------------------------------|------------------------------|
| num_class        | Integer         | RequiredN/A<br>for<br>Multiclass<br>classification. |                              |
| num_round        | Integer         | 100                                                 | N/A                          |
| tree_method      | String          | Auto                                                | N/A                          |
| max_depth        | Integer         | 6                                                   | [0 , 10]                     |
| min_child_weight | Float           | 1                                                   | MinValue: 0, MaxValue: 120   |
| subsample        | Float           | 1                                                   | MinValue: 0.5, MaxValue: 1   |
| gamma            | Float           | 0                                                   | MinValue: 0, MaxValue: 5     |
| alpha            | Float           | 0                                                   | MinValue: 0, MaxValue: 1000  |
| eta              | Float           | 0.3                                                 | MinValue: 0.1, MaxValue: 0.5 |

| Parameter name   | Parameter value | Default value | Notes                       |
|------------------|-----------------|---------------|-----------------------------|
| colsample_bytree | Float           | 1             | MinValue: 0.1, MaxValue: 1  |
| colsample_bytree | Float           | 1             | MinValue: 0.1, MaxValue: 1  |
| colsample_bytree | Float           | 1             | MinValue: 0.5, MaxValue: 1  |
| lambda           | Float           | 1             | MinValue: 0, MaxValue: 1000 |
| max_delta_step   | integer         | 0             | [0, 10]                     |

SETTINGS ( S3\_BUCKET 'bucket' | KMS\_KEY\_ID 'kms\_string' , | S3\_GARBAGE\_COLLECT on / off, | MAX\_CELLS integer , | MAX\_RUNTIME (...) )

S3\_BUCKET clause specifies the Amazon S3 location that is used to store intermediate results.

(Optional) KMS\_KEY\_ID specifies if Amazon Redshift uses server-side encryption with an AWS KMS key to protect data at rest. Data in transit is protected with Secure Sockets Layer (SSL).

(Optional) S3\_GARBAGE\_COLLECT { ON | OFF } specifies whether Amazon Redshift performs garbage collection on the resulting datasets used to train models and the models. If set to OFF, the resulting datasets used to train models and the models remains in Amazon S3 and can be used for other purposes. If set to ON, Amazon Redshift deletes the artifacts in Amazon S3 after the training completes. The default is ON.

(Optional) MAX\_CELLS specifies the number of cells in the training data. This value is the product of the number of records (in the training query or table) times the number of columns. The default is 1,000,000.

(Optional) MAX\_RUNTIME specifies the maximum amount of time to train. Training jobs often complete sooner depending on dataset size. This specifies the maximum amount of time the training should take. The default is 5,400 (90 minutes).

## Usage notes

When using CREATE MODEL, consider the following:

- The CREATE MODEL statement operates in an asynchronous mode and returns upon the export of training data to Amazon S3. The remaining steps of training in Amazon SageMaker occur in the background. While training is in progress, the corresponding inference function is visible but can't be run. You can query [STV\\_ML\\_MODEL\\_INFO \(p. 1587\)](#) to see the state of training.
- The training can run for up to 90 minutes in the background, by default in the Auto model and can be extended. To cancel the training, simply run the [DROP MODEL \(p. 868\)](#) command.
- The Amazon Redshift cluster that you use to create the model and the Amazon S3 bucket that is used to stage the training data and model artifacts must be in the same AWS Region.
- During the model training, Amazon Redshift and SageMaker store intermediate artifacts in the Amazon S3 bucket that you provide. By default, Amazon Redshift performs garbage collection at the end of the CREATE MODEL operation. Amazon Redshift removes those objects from Amazon S3. To retain those artifacts on Amazon S3, set the S3\_GARBAGE\_COLLECT OFF option.
- You must use at least 500 rows in the training data provided in the FROM clause.
- You can only specify up to 256 feature (input) columns in the FROM { table\_name | ( select\_query ) } clause when using the CREATE MODEL statement.
- For AUTO ON, the column types that you can use as the training set are SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE, BOOLEAN, CHAR, VARCHAR, DATE, TIME, TIMETZ, TIMESTAMP, and

TIMESTAMPTZ. For AUTO OFF, the column types that you can use as the training set are SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE, and BOOLEAN.

- You can't use DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPTZ, GEOMETRY, GEOGRAPHY, HLLSKETCH, SUPER, or VARBYTE as the target column type.
- To improve model accuracy, do one of the following:
  - Add as many relevant columns in the CREATE MODEL command as possible when you specify the training data in the FROM clause.
  - Use a larger value for MAX\_RUNTIME and MAX\_CELLS. Larger values for this parameter increase the cost of training a model.
- The CREATE MODEL statement execution returns as soon as the training data is computed and exported to the Amazon S3 bucket. After that point, you can check the status of the training using the SHOW MODEL command. When a model being trained in the background fails, you can check the error using SHOW MODEL. You can't retry a failed model. Use DROP MODEL to remove a failed model and recreate a new model. For more information about SHOW MODEL, see [SHOW MODEL \(p. 984\)](#).
- Local BYOM supports the same kind of models that Amazon Redshift ML supports for non-BYOM cases. Amazon Redshift supports plain XGBoost (using XGBoost version 1.0 or later), KMEANS models without preprocessors, and XGBOOST/MLP/Linear Learner models trained by trained by Amazon SageMaker Autopilot. It supports the latter with preprocessors that Autopilot has specified that are also supported by Amazon SageMaker Neo.
- If your Amazon Redshift cluster has enhanced routing enabled for your virtual private cloud (VPC), make sure to create an Amazon S3 VPC endpoint and an SageMaker VPC endpoint for the VPC that your cluster is in. Doing this enables the traffic to run through your VPC between these services during CREATE MODEL. For more information, see [SageMaker Clarify Job Amazon VPC Subnets and Security Groups](#).

## Use cases

The following use cases demonstrate how to use CREATE MODEL to suit your needs.

### Simple CREATE MODEL

The following summarizes the basic options of the CREATE MODEL syntax.

#### Simple CREATE MODEL syntax

```
CREATE MODEL model_name
FROM { table_name | (select_query) }
TARGET column_name
FUNCTION prediction_function_name
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
SETTINGS (
S3_BUCKET 'bucket',
[MAX_CELLS integer]
)
```

#### Simple CREATE MODEL parameters

##### *model\_name*

The name of the model. The model name in a schema must be unique.

##### FROM { *table\_name* | ( *select\_query* ) }

The *table\_name* or the query that specifies the training data. They can either be an existing table in the system, or an Amazon Redshift-compatible SELECT query enclosed with parentheses, that is (). There must be at least two columns in the query result.

**TARGET** *column\_name*

The name of the column that becomes the prediction target. The column must exist in the FROM clause.

**FUNCTION** *prediction\_function\_name*

A value that specifies the name of the Amazon Redshift machine learning function to be generated by the CREATE MODEL and used to make predictions using this model. The function is created in the same schema as the model object and can be overloaded.

Amazon Redshift machine learning supports models, such as Xtreme Gradient Boosted tree (XGBoost) models for regression and classification.

**IAM\_ROLE** { default | 'arn:aws:iam::<account-id>:role/<role-name>' }

Use the default keyword to have Amazon Redshift use the IAM role that is set as default and associated with the cluster when the CREAT MODEL command runs. Alternatively, you can specify the ARN of an IAM role to use that role.

**S3\_BUCKET** 'bucket'

The name of the Amazon S3 bucket that you previously created used to share training data and artifacts between Amazon Redshift and SageMaker. Amazon Redshift creates a subfolder in this bucket prior to unload of the training data. When training is complete, Amazon Redshift deletes the created subfolder and its contents.

**MAX\_CELLS** integer

The maximum number of cells to export from the FROM clause. The default is 1,000,000.

The number of cells is the product of the number of rows in the training data (produced by the FROM clause table or query) times the number of columns. If the number of cells in the training data are more than that specified by the max\_cells parameter, CREATE MODEL downsamples the FROM clause training data to reduce the size of the training set below MAX\_CELLS. Allowing larger training datasets can produce higher accuracy but also can mean the model takes longer to train and costs more.

For information about costs of using Amazon Redshift, see [Costs for using Amazon Redshift ML \(p. 399\)](#).

For more information about costs associated with various cell numbers and free trial details, see [Amazon Redshift pricing](#).

## CREATE MODEL with user guidance

Following, you can find a description of options for CREATE MODEL in addition to the options described in [Simple CREATE MODEL \(p. 805\)](#).

By default, CREATE MODEL searches for the best combination of preprocessing and model for your specific dataset. You might want additional control or introduce additional domain knowledge (such as problem type or objective) over your model. In a customer churn scenario, if the outcome "customer is not active" is rare, then the F1 objective is often preferred to the accuracy objective. Because high accuracy models might predict "customer is active" all the time, this results in high accuracy but little business value. For information about F1 objective, see [AutoMLJobObjective](#) in the *Amazon SageMaker API Reference*.

Then the CREATE MODEL follows your suggestions on the specified aspects, such as the objective. At the same time, the CREATE MODEL automatically discovers the best preprocessors and the best hyperparameters.

## [CREATE MODEL with user guidance syntax](#)

CREATE MODEL offers more flexibility on the aspects that you can specify and the aspects that Amazon Redshift automatically discovers.

```
CREATE MODEL model_name
FROM { table_name | (select_statement) }
TARGET column_name
FUNCTION function_name
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
[MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER }]
[PROBLEM_TYPE (REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION)]
[OBJECTIVE ('MSE' | 'Accuracy' | 'F1' | 'F1Macro' | 'AUC')]
SETTINGS (
S3_BUCKET 'bucket', |
S3_GARBAGE_COLLECT { ON | OFF }, |
KMS_KEY_ID 'kms_key_id', |
MAX_CELLS integer, |
MAX_RUNTIME integer (, ...)
)
```

## [CREATE MODEL with user guidance parameters](#)

*MODEL\_TYPE* { XGBOOST | MLP | LINEAR\_LEARNER }

(Optional) Specifies the model type. You can specify if you want to train a model of a specific model type, such as XGBoost, multilayer perceptron (MLP), or Linear Learner, which are all algorithms that Amazon SageMaker Autopilot supports. If you don't specify the parameter, then all supported model types are searched during training for the best model.

*PROBLEM\_TYPE* ( REGRESSION | BINARY\_CLASSIFICATION | MULTICLASS\_CLASSIFICATION )

(Optional) Specifies the problem type. If you know the problem type, you can restrict Amazon Redshift to only search of the best model of that specific model type. If you don't specify this parameter, a problem type is discovered during the training, based on your data.

*OBJECTIVE* ( 'MSE' | 'Accuracy' | 'F1' | 'F1Macro' | 'AUC' )

(Optional) Specifies the name of the objective metric used to measure the predictive quality of a machine learning system. This metric is optimized during training to provide the best estimate for model parameter values from data. If you don't specify a metric explicitly, the default behavior is to automatically use MSE: for regression, F1: for binary classification, Accuracy: for multiclass classification. For more information about objectives, see [AutoMLJobObjective](#) in the *Amazon SageMaker API Reference*.

*MAX\_CELLS* *integer*

(Optional) Specifies the number of cells in the training data. This value is the product of the number of records (in the training query or table) times the number of columns. The default is 1,000,000.

*MAX\_RUNTIME* *integer*

(Optional) Specifies the maximum amount of time to train. Training jobs often complete sooner depending on dataset size. This specifies the maximum amount of time the training should take. The default is 5,400 (90 minutes).

*S3\_GARBAGE\_COLLECT* { ON | OFF }

(Optional) Specifies whether Amazon Redshift performs garbage collection on the resulting datasets used to train models and the models. If set to OFF, the resulting datasets used to train models and the models remains in Amazon S3 and can be used for other purposes. If set to ON, Amazon Redshift deletes the artifacts in Amazon S3 after the training completes. The default is ON.

**KMS\_KEY\_ID 'kms\_key\_id'**

(Optional) Specifies if Amazon Redshift uses server-side encryption with an AWS KMS key to protect data at rest. Data in transit is protected with Secure Sockets Layer (SSL).

**PREPROCESSORS 'string'**

(Optional) Specifies certain combinations of preprocessors to certain sets of columns. The format is a list of columnSets, and the appropriate transforms to be applied to each set of columns.

Amazon Redshift applies all the transformers in a specific transformers list to all columns in the corresponding ColumnSet. For example, to apply OneHotEncoder with Imputer to columns t1 and t2, use the sample command following.

```

CREATE MODEL customer_churn
FROM customer_data
TARGET 'Churn'
FUNCTION predict_churn
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
PROBLEM_TYPE BINARY_CLASSIFICATION
OBJECTIVE 'F1'
PREPROCESSORS '[

 ...

 {"ColumnSet": [
 "t1",
 "t2"
],
 "Transformers": [
 "OneHotEncoder",
 "Imputer"
]
},
 {"ColumnSet": [
 "t3"
],
 "Transformers": [
 "OneHotEncoder"
]
},
 {"ColumnSet": [
 "temp"
],
 "Transformers": [
 "Imputer",
 "NumericPassthrough"
]
}
]'

SETTINGS (
S3_BUCKET 'bucket'
)

```

Amazon Redshift supports the following transformers:

- **OneHotEncoder** – Typically used to encode a discrete value into a binary vector with one nonzero value. This transformer is suitable for many machine learning models.
- **OrdinalEncoder** – Encodes discrete values into a single integer. This transformer is suitable for certain machine learning models, such as MLP and Linear Learner.
- **NumericPassthrough** – Passes input as is into the model.
- **Imputer** – Fills in missing values and not a number (NaN) values.
- **ImputerWithIndicator** – Fills in missing values and NaN values. This transformer also creates an indicator of whether any values were missing and filled in.

- Normalizer – Normalizes values, which can improve the performance of many machine learning algorithms.
- DateTimeVectorizer – Creates a vector embedding, representing a column of datetime data type that can be used in machine learning models.
- PCA – Projects the data into a lower dimensional space to reduce the number of features while keeping as much information as possible.
- StandardScaler – Standardizes features by removing the mean and scaling to unit variance.
- MinMax – Transforms features by scaling each feature to a given range.

Amazon Redshift ML stores the trained transformers, and automatically applies them as part of the prediction query. You don't need to specify them when generating predictions from your model.

## CREATE XGBoost models with AUTO OFF

The AUTO OFF CREATE MODEL has generally different objectives from the default CREATE MODEL.

As an advanced user who already knows the model type that you want and hyperparameters to use when training these models, you can use CREATE MODEL with AUTO OFF to turn off the CREATE MODEL automatic discovery of preprocessors and hyperparameters. To do so, you explicitly specify the model type. XGBoost is currently the only model type supported when AUTO is set to OFF. You can specify hyperparameters. Amazon Redshift uses default values for any hyperparameters that you specified.

### CREATE XGBoost models with AUTO OFF syntax

```
CREATE MODEL model_name
FROM { table_name | (select_statement) }
TARGET column_name
FUNCTION function_name
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
AUTO OFF
MODEL_TYPE XGBOOST
OBJECTIVE { 'reg:squarederror' | 'reg:squaredlogerror' | 'reg:logistic' |
 'reg:pseudohubererror' | 'reg:tweedie' | 'binary:logistic' | 'binary:hinge' |
 'multi:softmax' | 'rank:pairwise' | 'rank:ndcg' }
HYPERPARAMETERS DEFAULT EXCEPT (
 NUM_ROUND '10',
 ETA '0.2',
 NUM_CLASS '10',
 (, ...)
)
PREPROCESSORS 'none'
SETTINGS (
 S3_BUCKET 'bucket', |
 S3_GARBAGE_COLLECT { ON | OFF }, |
 KMS_KEY_ID 'kms_key_id', |
 MAX_CELLS integer, |
 MAX_RUNTIME integer (, ...)
)
```

### CREATE XGBoost models with AUTO OFF parameters

#### AUTO OFF

Turns off CREATE MODEL automatic discovery of preprocessor, algorithm, and hyper-parameters selection.

#### MODEL\_TYPE XGBOOST

Specifies to use XGBOOST to train the model.

### OBJECTIVE str

Specifies an objective recognized by the algorithm. Amazon Redshift supports reg:squarederror, reg:squaredlogerror, reg:logistic, reg:pseudohubererror, reg:tweedie, binary:logistic, binary:hinge, multi:softmax. For more information about these objectives, see [Learning task parameters](#) in the XGBoost documentation.

### HYPERPARAMETERS { DEFAULT | DEFAULT EXCEPT ( key 'value' (...) ) }

Specifies whether the default XGBoost parameters are used or overridden by user-specified values. The values must be enclosed with single quotes. Following are examples of parameters for XGBoost and their defaults.

| Parameter name   | Parameter value | Default value                                       | Notes                        |
|------------------|-----------------|-----------------------------------------------------|------------------------------|
| num_class        | Integer         | RequiredN/A<br>for<br>Multiclass<br>classification. |                              |
| num_round        | Integer         | 100                                                 | N/A                          |
| tree_method      | String          | Auto                                                | N/A                          |
| max_depth        | Integer         | 6                                                   | [0 , 10]                     |
| min_child_weight | Float           | 1                                                   | MinValue: 0, MaxValue: 120   |
| subsample        | Float           | 1                                                   | MinValue: 0.5, MaxValue: 1   |
| gamma            | Float           | 0                                                   | MinValue: 0, MaxValue: 5     |
| alpha            | Float           | 0                                                   | MinValue: 0, MaxValue: 1000  |
| eta              | Float           | 0.3                                                 | MinValue: 0.1, MaxValue: 0.5 |
| colsample_bytree | Float           | 1                                                   | MinValue: 0.1, MaxValue: 1   |
| colsample_bycol  | Float           | 1                                                   | MinValue: 0.1, MaxValue: 1   |
| colsample_bybin  | Float           | 1                                                   | MinValue: 0.5, MaxValue: 1   |
| lambda           | Float           | 1                                                   | MinValue: 0, MaxValue: 1000  |
| max_delta_step   | Integer         | 0                                                   | [0, 10]                      |

The following example prepares data for XGBoost.

```
DROP TABLE IF EXISTS abalone_xgb;

CREATE TABLE abalone_xgb (
 length_val float,
 diameter float,
 height float,
 whole_weight float,
 shucked_weight float,
 viscera_weight float,
 shell_weight float,
 rings int,
 record_number int);
```

```
COPY abalone_xgb
FROM 's3://redshift-downloads/redshift-ml/abalone_xg/'
REGION 'us-east-1'
IAM_ROLE default
IGNOREHEADER 1 CSV;
```

The following example creates an XGBoost model with specified advanced options, such as MODEL\_TYPE, OBJECTIVE, and PREPROCESSORS.

```
DROP MODEL abalone_xgboost_multi_predict_age;

CREATE MODEL abalone_xgboost_multi_predict_age
FROM (SELECT length_val,
 diameter,
 height,
 whole_weight,
 shucked_weight,
 viscera_weight,
 shell_weight,
 rings
 FROM abalone_xgb WHERE record_number < 2500)
TARGET rings FUNCTION ml_fn_abalone_xgboost_multi_predict_age
IAM_ROLE default
AUTO OFF
MODEL_TYPE XGBOOST
OBJECTIVE 'multi:softmax'
PREPROCESSORS 'none'
HYPERPARAMETERS DEFAULT EXCEPT (NUM_ROUND '100', NUM_CLASS '30')
SETTINGS (S3_BUCKET 'your-bucket');
```

The following example uses an inference query to predict the age of the fish with a record number greater than 2500. It uses the function `ml_fn_abalone_xgboost_multi_predict_age` created from the above command.

```
select ml_fn_abalone_xgboost_multi_predict_age(length_val,
 diameter,
 height,
 whole_weight,
 shucked_weight,
 viscera_weight,
 shell_weight)+1.5 as age
from abalone_xgb where record_number > 2500;
```

## Bring your own model (BYOM) - local inference

Amazon Redshift ML supports using bring your own model (BYOM) for local inference.

The following summarizes the options for the CREATE MODEL syntax for BYOM. You can use a model trained outside of Amazon Redshift with Amazon SageMaker for in-database inference locally in Amazon Redshift.

### [CREATE MODEL syntax for local inference](#)

The following describes the CREATE MODEL syntax for local inference.

```
CREATE MODEL model_name
FROM ('job_name' | 's3_path')
FUNCTION function_name (data_type [, ...])
RETURNS data_type
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
```

```
[SETTINGS (
 S3_BUCKET 'bucket', | --required
 KMS_KEY_ID 'kms_string') --optional
];
```

Amazon Redshift currently only supports pretrained XGBoost, MLP, and Linear Learner models for BYOM. You can import SageMaker Autopilot and models directly trained in Amazon SageMaker for local inference using this path.

#### [CREATE MODEL parameters for local inference](#)

*model\_name*

The name of the model. The model name in a schema must be unique.

FROM ('*job\_name*' | 's3\_path')

The *job\_name* uses an Amazon SageMaker job name as the input. The job name can either be an Amazon SageMaker training job name or an Amazon SageMaker Autopilot job name. The job must be created in the same AWS account that owns the Amazon Redshift cluster. To find the job name, launch Amazon SageMaker. In the **Training** dropdown menu, choose **Training jobs**.

The 's3\_path' specifies the S3 location of the .tar.gz model artifacts file that is to be used when creating the model.

FUNCTION *function\_name* (*data\_type* [, ...])

The name of the function to be created and the data types of the input arguments. You can provide a schema name.

RETURNS *data\_type*

The data type of the value returned by the function.

IAM\_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>'}

Use the default keyword to have Amazon Redshift use the IAM role that is set as default and associated with the cluster when the CREATE MODEL command runs.

Use the Amazon Resource Name (ARN) for an IAM role that your cluster uses for authentication and authorization.

SETTINGS ( S3\_BUCKET 'bucket', | KMS\_KEY\_ID 'kms\_string' )

The S3\_BUCKET clause specifies the Amazon S3 location that is used to store intermediate results.

(Optional) The KMS\_KEY\_ID clause specifies if Amazon Redshift uses server-side encryption with an AWS KMS key to protect data at rest. Data in transit is protected with Secure Sockets Layer (SSL).

For more information, see [CREATE MODEL with user guidance \(p. 806\)](#).

#### [CREATE MODEL for local inference example](#)

The following example creates a model that has been previously trained in Amazon SageMaker, outside of Amazon Redshift. Because the model type is supported by Amazon Redshift ML for local inference, the following CREATE MODEL creates a function that can be used locally in Amazon Redshift. You can provide a SageMaker training job name.

```
CREATE MODEL customer_churn
FROM 'training-job-customer-churn-v4'
FUNCTION customer_churn_predict (varchar, int, float, float)
RETURNS int
IAM_ROLE default
```

```
SETTINGS (S3_BUCKET 'your-bucket');
```

After the model is created, you can use the function *customer\_churn\_predict* with the specified argument types to make predictions.

## Bring your own model (BYOM) - remote inference

Amazon Redshift ML also supports using bring your own model (BYOM) for remote inference.

The following summarizes the options for the CREATE MODEL syntax for BYOM.

### CREATE MODEL syntax for remote inference

The following describes the CREATE MODEL syntax for remote inference.

```
CREATE MODEL model_name
FUNCTION function_name (data_type [, ...])
RETURNS data_type
SAGEMAKER 'endpoint_name'[:'model_name']
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' };
```

### CREATE MODEL parameters for remote inference

#### *model\_name*

The name of the model. The model name in a schema must be unique.

#### FUNCTION *fn\_name* ( [*data\_type*] [, ...] )

The name of the function and the data types of the input arguments. You can provide a schema name.

#### RETURNS *data\_type*

The data type of the value returned by the function.

#### SAGEMAKER '*endpoint\_name*'[:'*model\_name*']

The name of the Amazon SageMaker endpoint. If the endpoint name points to a multimodel endpoint, add the name of the model to use. The endpoint must be hosted in the same AWS Region as the Amazon Redshift cluster. To find your endpoint, launch Amazon SageMaker. In the **Inference** dropdown menu, choose **Endpoints**.

#### IAM\_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }

Use the default keyword to have Amazon Redshift use the IAM role that is set as default and associated with the cluster when the CREATE MODEL command runs. Alternatively, you can specify the ARN of an IAM role to use that role.

When the model is deployed to a SageMaker endpoint, SageMaker creates the information of the model in Amazon Redshift. It then performs inference through the external function. You can use the SHOW MODEL command to view the model information on your Amazon Redshift cluster.

### CREATE MODEL for remote inference usage notes

Before using CREATE MODEL for remote inference, consider the following:

- The model must accept inputs in the format of comma-separated values (CSV) through a content type of text or CSV in SageMaker.
- The endpoint must be hosted by the same AWS account that owns the Amazon Redshift cluster.
- The outputs of models must be a single value of the type specified on creating the function, in the format of comma-separated values (CSV) through a content type of text or CSV in SageMaker.

- Models accept nulls as empty strings.
- Make sure either that the Amazon SageMaker endpoint has enough resources to accommodate inference calls from Amazon Redshift or that the Amazon SageMaker endpoint can be automatically scaled.

### [CREATE MODEL for remote inference example](#)

The following example creates a model that uses a SageMaker endpoint to make predictions. Make sure that the endpoint is running to make predictions and specify its name in the CREATE MODEL command.

```
CREATE MODEL remote_customer_churn
FUNCTION remote_fn_customer_churn_predict (varchar, int, float, float)
RETURNS int
SAGEMAKER 'customer-churn-endpoint'
IAM_ROLE default;
```

### [CREATE MODEL with K-MEANS](#)

Amazon Redshift supports the K-Means algorithm that groups data that isn't labeled. This algorithm solves clustering problems where you want to discover groupings in the data. Unclassified data is grouped and partitioned based on its similarities and differences.

#### [CREATE MODEL with K-MEANS syntax](#)

```
CREATE MODEL model_name
FROM { table_name | (select_statement) }
FUNCTION function_name
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
AUTO OFF
MODEL_TYPE KMEANS
PREPROCESSORS 'string'
HYPERPARAMETERS DEFAULT EXCEPT (K 'val' [, ...])
SETTINGS (
S3_BUCKET 'bucket',
KMS_KEY_ID 'kms_string', |
-- optional
S3_GARBAGE_COLLECT on / off, |
-- optional
MAX_CELLS integer, |
-- optional
MAX_RUNTIME integer
-- optional);
```

#### [CREATE MODEL with K-MEANS parameters](#)

##### **AUTO OFF**

Turns off CREATE MODEL automatic discovery of preprocessor, algorithm, and hyper-parameters selection.

##### **MODEL\_TYPE KMEANS**

Specifies to use KMEANS to train the model.

##### **PREPROCESSORS 'string'**

Specifies certain combinations of preprocessors to certain sets of columns. The format is a list of columnSets, and the appropriate transforms to be applied to each set of columns. Amazon Redshift supports 3 K-Means preprocessors, namely StandardScaler, MinMax, and NumericPassthrough. If you don't want to apply any preprocessing for K-Means, choose NumericPassthrough explicitly as

a transformer. For more information about supported transformers, see [CREATE MODEL with user guidance parameters \(p. 807\)](#).

The K-Means algorithm uses Euclidean distance to calculate similarity. Preprocessing the data ensures that the features of the model stay on the same scale and produce reliable results.

HYPERPARAMETERS DEFAULT EXCEPT ( K 'val' [, ...] )

Specifies whether the K-Means parameters are used. You must specify the K parameter when using the K-Means algorithm. For more information, see [K-Means Hyperparameters](#) in the *Amazon SageMaker Developer Guide*

The following example prepares data for K-Means.

```
CREATE MODEL customers_clusters
FROM customers
FUNCTION customers_cluster
IAM_ROLE default
AUTO OFF
MODEL_TYPE KMEANS
PREPROCESSORS '[

{
 "ColumnSet": ["*"],
 "Transformers": ["NumericPassthrough"]
}
]'

HYPERPARAMETERS DEFAULT EXCEPT (K '5')
SETTINGS (S3_BUCKET 'bucket');

select customer_id, customers_cluster(...) from customers;
customer_id | customers_cluster

12345 1
12346 2
12347 4
12348 0
```

## CREATE PROCEDURE

Creates a new stored procedure or replaces an existing procedure for the current database.

### Required privileges

Following are required privileges for CREATE PROCEDURE:

- For CREATE PROCEDURE:
  - Superuser
  - Users with the CREATE [ OR REPLACE ] PROCEDURE privilege
- For REPLACE PROCEDURE:
  - Superuser
  - Users with the CREATE [ OR REPLACE ] PROCEDURE privilege
  - Procedure owner

### Syntax

```
CREATE [OR REPLACE] PROCEDURE sp_procedure_name
```

```
([[argname] [argmode] argtype [, ...]])
[NONATOMIC]
AS $$
procedure_body
$$ LANGUAGE plpgsql
[{ SECURITY INVOKER | SECURITY DEFINER }]
[SET configuration_parameter { TO value | = value }]
```

## Parameters

### OR REPLACE

A clause that specifies that if a procedure with the same name and input argument data types, or signature, as this one already exists, the existing procedure is replaced. You can only replace a procedure with a new procedure that defines an identical set of data types. You must be a superuser or the owner to replace a procedure.

If you define a procedure with the same name as an existing procedure, but a different signature, you create a new procedure. In other words, the procedure name is overloaded. For more information, see [Overloading procedure names \(p. 166\)](#).

### *sp\_procedure\_name*

The name of the procedure. If you specify a schema name (such as `myschema.myprocedure`), the procedure is created in the specified schema. Otherwise, the procedure is created in the current schema. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

We recommend that you prefix all stored procedure names with `sp_`. Amazon Redshift reserves the `sp_` prefix for stored procedure names. By using the `sp_` prefix, you ensure that your stored procedure name doesn't conflict with any existing or future Amazon Redshift built-in stored procedure or function names. For more information, see [Naming stored procedures \(p. 166\)](#).

You can define more than one procedure with the same name if the data types for the input arguments, or signatures, are different. In other words, in this case the procedure name is overloaded. For more information, see [Overloading procedure names \(p. 166\)](#)

### *[argname] [ argmode] argtype*

A list of argument names, argument modes, and data types. Only the data type is required. Name and mode are optional and their position can be switched.

The argument mode can be IN, OUT, or INOUT. The default is IN.

You can use OUT and INOUT arguments to return one or more values from a procedure call. When there are OUT or INOUT arguments, the procedure call returns one result row containing *n* columns, where *n* is the total number of OUT or INOUT arguments.

INOUT arguments are input and output arguments at the same time. *Input arguments* include both IN and INOUT arguments, and *output arguments* include both OUT and INOUT arguments.

OUT arguments aren't specified as part of the CALL statement. Specify INOUT arguments in the stored procedure CALL statement. INOUT arguments can be useful when passing and returning values from a nested call, and also when returning a `refcursor`. For more information on `refcursor` types, see [Cursors \(p. 195\)](#).

The argument data types can be any standard Amazon Redshift data type. In addition, an argument data type can be `refcursor`.

You can specify a maximum of 32 input arguments and 32 output arguments.

### AS \$\$ *procedure\_body* \$\$

A construct that encloses the procedure to be run. The literal keywords AS \$\$ and \$\$ are required.

Amazon Redshift requires you to enclose the statement in your procedure by using a format called dollar quoting. Anything within the enclosure is passed exactly as is. You don't need to escape any special characters because the contents of the string are written literally.

With *dollar quoting*, you use a pair of dollar signs (\$\$) to signify the start and the end of the statement to run, as shown in the following example.

```
$$ my statement $$
```

Optionally, between the dollar signs in each pair, you can specify a string to help identify the statement. The string that you use must be the same in both the start and the end of the enclosure pairs. This string is case-sensitive, and it follows the same constraints as an unquoted identifier except that it can't contain dollar signs. The following example uses the string test.

```
$test$ my statement $test$
```

This syntax is also useful for nested dollar quoting. For more information about dollar quoting, see "Dollar-quoted String Constants" under [Lexical Structure](#) in the PostgreSQL documentation.

#### *procedure\_body*

A set of valid PL/pgSQL statements. PL/pgSQL statements augment SQL commands with procedural constructs, including looping and conditional expressions, to control logical flow. Most SQL commands can be used in the procedure body, including data modification language (DML) such as COPY, UNLOAD and INSERT, and data definition language (DDL) such as CREATE TABLE. For more information, see [PL/pgSQL language reference \(p. 182\)](#).

#### *LANGUAGE plpgsql*

A language value. Specify plpgsql. You must have permission for usage on language to use plpgsql. For more information, see [GRANT \(p. 887\)](#).

#### *NONATOMIC*

Creates the stored procedure in a nonatomic transaction mode. NONATOMIC mode automatically commits the statements inside the procedure. Additionally, when an error occurs inside the NONATOMIC procedure, the error is not re-thrown if it is handled by an exception block. For more information, see [Managing transactions \(p. 169\)](#) and [RAISE \(p. 196\)](#).

When you define a stored procedure as NONATOMIC, consider the following:

- When you nest stored procedure calls, all the procedures must be created in the same transaction mode.
- The SECURITY DEFINER option and SET configuration\_parameter option are not supported when creating a procedure in NONATOMIC mode.
- Any cursor that is opened (explicitly or implicitly) is closed automatically when an implicit commit is processed. Therefore, you must open an explicit transaction before beginning a cursor loop to ensure that any SQL within the loop's iteration is not implicitly committed.

#### *SECURITY INVOKER | SECURITY DEFINER*

The SECURITY DEFINER option is not supported when NONATOMIC is specified.

The security mode for the procedure determines the procedure's access privileges at runtime. The procedure must have permission to access the underlying database objects.

For SECURITY INVOKER mode, the procedure uses the privileges of the user calling the procedure. The user must have explicit permissions on the underlying database objects. The default is SECURITY INVOKER.

For SECURITY DEFINER mode, the procedure uses the privileges of the procedure owner. The procedure owner is defined as the user that owns the procedure at run time, not necessarily the user that initially defined the procedure. The user calling the procedure needs execute privilege on the procedure, but doesn't need any privileges on the underlying objects.

SET configuration\_parameter { TO value | = value }

These options are not supported when NONATOMIC is specified.

The SET clause causes the specified configuration\_parameter to be set to the specified value when the procedure is entered. This clause then restores configuration\_parameter to its earlier value when the procedure exits.

## Usage notes

The NONATOMIC transaction mode is not supported when the stored procedure is created using JDBC or ODBC drivers.

## Examples

### Note

If when running these examples you encounter an error similar to:

```
ERROR: 42601: [Amazon](500310) unterminated dollar-quoted string at or near "$$"
```

See [Overview of stored procedures in Amazon Redshift \(p. 164\)](#).

The following example creates a procedure with two input parameters.

```
CREATE OR REPLACE PROCEDURE test_sp1(f1 int, f2 varchar(20))
AS $$
DECLARE
 min_val int;
BEGIN
 DROP TABLE IF EXISTS tmp_tbl;
 CREATE TEMP TABLE tmp_tbl(id int);
 INSERT INTO tmp_tbl values (f1),(10001),(10002);
 SELECT INTO min_val MIN(id) FROM tmp_tbl;
 RAISE INFO 'min_val = %, f2 = %', min_val, f2;
END;
$$ LANGUAGE plpgsql;
```

### Note

When you write stored procedures, we recommend a best practice for securing sensitive values: Don't hard code any sensitive information in stored procedure logic. For example, don't assign a user password in a CREATE USER statement in the body of a stored procedure. This poses a security risk, because hard-coded values can be recorded as schema metadata in catalog tables. Instead, pass sensitive values, such as passwords, as arguments to the stored procedure, by means of parameters.

For more information about stored procedures, see [CREATE PROCEDURE](#) and [Creating stored procedures in Amazon Redshift](#). For more information about catalog tables, see [System catalog tables](#).

The following example creates a procedure with one IN parameter, one OUT parameter, and one INOUT parameter.

```
CREATE OR REPLACE PROCEDURE test_sp2(f1 IN int, f2 INOUT varchar(256), out_var OUT
varchar(256))
```

```

AS $$

DECLARE
 loop_var int;
BEGIN
 IF f1 is null OR f2 is null THEN
 RAISE EXCEPTION 'input cannot be null';
 END IF;
 DROP TABLE if exists my_etl;
 CREATE TEMP TABLE my_etl(a int, b varchar);
 FOR loop_var IN 1..f1 LOOP
 insert into my_etl values (loop_var, f2);
 f2 := f2 || '+' || f2;
 END LOOP;
 SELECT INTO out_var count(*) from my_etl;
END;
$$ LANGUAGE plpgsql;

```

## CREATE RLS POLICY

Creates a new row-level security policy to provide granular access to database objects.

Superusers and users or roles that have the sys:secadmin role can create a policy.

### Syntax

```

CREATE RLS POLICY policy_name
[WITH (column_name data_type [, ...]) [[AS] relation_alias]]
USING (using_predicate_exp)

```

### Parameters

*policy\_name*

The name of the policy.

**WITH** (*column\_name data\_type [, ...]*)

Specifies the *column\_name* and *data\_type* referenced to the columns of tables to which the policy is attached.

You can omit the WITH clause only when the RLS policy doesn't reference any columns of tables to which the policy is attached.

**AS** *relation\_alias*

Specifies an optional alias for the table that the RLS policy will be attached to.

**USING** ( *using\_predicate\_exp* )

Specifies a filter that is applied to the WHERE clause of the query. Amazon Redshift applies a policy predicate before the query-level user predicates. For example, **current\_user = 'joe'** and **price > 10** limits Joe to see only records with the price greater than \$10.

### Usage notes

When working with the CREATE RLS POLICY statement, observe the following:

- Amazon Redshift supports filters that can be part of a WHERE clause of a query.

- All policies being attached to a table must have been created with the same table alias.
- You don't require SELECT permission on lookup tables. When you create a policy, Amazon Redshift grants the SELECT permission on the lookup table for the respective policy. A lookup table is a table object used inside a policy definition.
- Amazon Redshift row-level security doesn't support the following object types inside a policy definition: catalog tables, cross-database relations, external tables, regular views, late-binding views, tables with RLS policies turned on, and temporary tables.

## Examples

The following SQL statements create the tables, users, and roles for the CREATE RLS POLICY example.

```
-- Create users and roles reference in the policy statements.
CREATE ROLE analyst;

CREATE ROLE consumer;

CREATE USER bob WITH PASSWORD 'Name_is_bob_1';

CREATE USER alice WITH PASSWORD 'Name_is_alice_1';

CREATE USER joe WITH PASSWORD 'Name_is_joe_1';

GRANT ROLE sys:secadmin TO bob;

GRANT ROLE analyst TO alice;

GRANT ROLE consumer TO joe;

GRANT ALL ON TABLE ticket_category_redshift TO PUBLIC;
```

The following example creates a policy called policy\_concerts.

```
CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Concerts');
```

## CREATE ROLE

Creates a new custom role that is a collection of permissions. For a list of Amazon Redshift system-defined roles, see [the section called "Amazon Redshift system-defined roles" \(p. 542\)](#).

### Required permissions

Following are the required privileges for CREATE ROLE.

- Superuser
- Users with the CREATE ROLE privilege

## Syntax

```
CREATE ROLE role_name
[EXTERNALID external_id]
```

## Parameters

### *role\_name*

The name of the role. The role name must be unique and can't be the same as any user names. A role name can't be a reserved word.

A superuser or regular user with the CREATE ROLE privilege can create roles. A user that is not a superuser but that has been granted USAGE to the role WITH GRANT OPTION and ALTER privilege can grant this role to anyone.

### EXTERNALID *external\_id*

The identifier for the role, which is associated with an identity provider. For more information, see [Native identity provider \(IdP\) federation for Amazon Redshift](#).

## Examples

The following example creates a role `sample_role1`.

```
CREATE ROLE sample_role1;
```

The following example creates a role `sample_role1`, with an external ID that is associated with an identity provider.

```
CREATE ROLE sample_role1 EXTERNALID "ABC123";
```

## CREATE SCHEMA

Defines a new schema for the current database.

## Required privileges

Following are required privileges for CREATE SCHEMA:

- Superuser
- Users with the CREATE SCHEMA privilege

## Syntax

```
CREATE SCHEMA [IF NOT EXISTS] schema_name [AUTHORIZATION username]
 [QUOTA {quota [MB | GB | TB] | UNLIMITED}] [schema_element [...]]
CREATE SCHEMA AUTHORIZATION username[QUOTA {quota [MB | GB | TB] | UNLIMITED}]
 [schema_element [...]]
```

## Parameters

### IF NOT EXISTS

Clause that indicates that if the specified schema already exists, the command should make no changes and return a message that the schema exists, rather than terminating with an error.

This clause is useful when scripting, so the script doesn't fail if CREATE SCHEMA tries to create a schema that already exists.

*schema\_name*

Name of the new schema. The schema name can't be PUBLIC. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

**Note**

The list of schemas in the [search\\_path \(p. 1696\)](#) configuration parameter determines the precedence of identically named objects when they are referenced without schema names.

AUTHORIZATION

Clause that gives ownership to a specified user.

*username*

Name of the schema owner.

*schema\_element*

Definition for one or more objects to be created within the schema.

QUOTA

The maximum amount of disk space that the specified schema can use. This space is the collective disk usage. It includes all permanent tables, materialized views under the specified schema, and duplicate copies of all tables with ALL distribution on each compute node. The schema quota doesn't take into account temporary tables created as part of a temporary namespace or schema.

To view the configured schema quotas, see [SVV\\_SCHEMA\\_QUOTA\\_STATE \(p. 1454\)](#).

To view the records where schema quotas were exceeded, see [STL\\_SCHEMA\\_QUOTA\\_VIOLATIONS \(p. 1557\)](#).

Amazon Redshift converts the selected value to megabytes. Gigabytes is the default unit of measurement when you don't specify a value.

You must be a database superuser to set and change a schema quota. A user that is not a superuser but that has CREATE SCHEMA permission can create a schema with a defined quota. When you create a schema without defining a quota, the schema has an unlimited quota. When you set the quota below the current value used by the schema, Amazon Redshift doesn't allow further ingestion until you free disk space. A DELETE statement deletes data from a table and disk space is freed up only when VACUUM runs.

Amazon Redshift checks each transaction for quota violations before committing the transaction. Amazon Redshift checks the size (the disk space used by all tables in a schema) of each modified schema against the set quota. Because the quota violation check occurs at the end of a transaction, the size limit can exceed the quota temporarily within a transaction before it's committed. When a transaction exceeds the quota, Amazon Redshift stops the transaction, prohibits subsequent ingestions, and reverts all the changes until you free disk space. Due to background VACUUM and internal cleanup, it is possible that a schema isn't full by the time that you check the schema after a canceled transaction.

As an exception, Amazon Redshift disregards the quota violation and commits transactions in certain cases. Amazon Redshift does this for transactions that consist solely of one or more of the following statements where there isn't an INSERT or COPY ingestion statement in the same transaction:

- DELETE
- TRUNCATE

- VACUUM
- DROP TABLE
- ALTER TABLE APPEND only when moving data from the full schema to another non-full schema

**UNLIMITED**

Amazon Redshift imposes no limit to the growth of the total size of the schema.

## Limits

Amazon Redshift enforces the following limits for schemas.

- There is a maximum of 9900 schemas per database.

## Examples

The following example creates a schema named US\_SALES and gives ownership to the user DWUSER.

```
create schema us_sales authorization dwuser;
```

The following example creates a schema named US\_SALES, gives ownership to the user DWUSER, and sets the quota to 50 GB.

```
create schema us_sales authorization dwuser QUOTA 50 GB;
```

To view the new schema, query the PG\_NAMESPACE catalog table as shown following.

```
select nspname as schema, usename as owner
from pg_namespace, pg_user
where pg_namespace.nspowner = pg_user.usessysid
and pg_user.usename ='dwuser';

schema | owner
-----+-----
us_sales | dwuser
(1 row)
```

The following example either creates the US\_SALES schema, or does nothing and returns a message if it already exists.

```
create schema if not exists us_sales;
```

## CREATE TABLE

Creates a new table in the current database. You define a list of columns, which each hold data of a distinct type. The owner of the table is the issuer of the CREATE TABLE command.

## Required privileges

Following are required privileges for CREATE TABLE:

- Superuser

- Users with the CREATE TABLE privilege

## Syntax

```

CREATE [[LOCAL] { TEMPORARY | TEMP }] TABLE
[IF NOT EXISTS] table_name
({ column_name data_type [column_attributes] [column_constraints]
| table_constraints
| LIKE parent_table [{ INCLUDING | EXCLUDING } DEFAULTS] }
[, ...])
[BACKUP { YES | NO }]
[table_attributes]

where column_attributes are:
[DEFAULT default_expr]
[IDENTITY (seed, step)]
[GENERATED BY DEFAULT AS IDENTITY (seed, step)]
[ENCODE encoding]
[DISTKEY]
[SORTKEY]
[COLLATE CASE_SENSITIVE | COLLATE CASE_INSENSITIVE]

and column_constraints are:
[{ NOT NULL | NULL }]
[{ UNIQUE | PRIMARY KEY }]
[REFERENCES reftable [(refcolumn)]]

and table_constraints are:
[UNIQUE (column_name [, ...])]
[PRIMARY KEY (column_name [, ...])]
[FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn)]]

and table_attributes are:
[DISTSTYLE { AUTO | EVEN | KEY | ALL }]
[DISTKEY (column_name)]
[[COMPOUND | INTERLEAVED] SORTKEY (column_name [,....]) | [SORTKEY AUTO]]
[ENCODE AUTO]

```

## Parameters

### LOCAL

Optional. Although this keyword is accepted in the statement, it has no effect in Amazon Redshift.

### TEMPORARY | TEMP

Keyword that creates a temporary table that is visible only within the current session. The table is automatically dropped at the end of the session in which it is created. The temporary table can have the same name as a permanent table. The temporary table is created in a separate, session-specific schema. (You can't specify a name for this schema.) This temporary schema becomes the first schema in the search path, so the temporary table takes precedence over the permanent table unless you qualify the table name with the schema name to access the permanent table. For more information about schemas and precedence, see [search\\_path \(p. 1696\)](#).

#### Note

By default, database users have permission to create temporary tables by their automatic membership in the PUBLIC group. To deny this privilege to a user, revoke the TEMP privilege from the PUBLIC group, and then explicitly grant the TEMP privilege only to specific users or groups of users.

#### IF NOT EXISTS

Clause that indicates that if the specified table already exists, the command should make no changes and return a message that the table exists, rather than stopping with an error. Note that the existing table might be nothing like the one that would have been created; only the table name is compared.

This clause is useful when scripting, so the script doesn't fail if CREATE TABLE tries to create a table that already exists.

#### *table\_name*

Name of the table to be created.

##### **Important**

If you specify a table name that begins with '#', the table is created as a temporary table.

The following is an example:

```
create table #newtable (id int);
```

You also reference the table with the '#'. For example:

```
select * from #newtable;
```

The maximum length for the table name is 127 bytes; longer names are truncated to 127 bytes. You can use UTF-8 multibyte characters up to a maximum of four bytes. Amazon Redshift enforces a quota of the number of tables per cluster by node type, including user-defined temporary tables and temporary tables created by Amazon Redshift during query processing or system maintenance. Optionally, the table name can be qualified with the database and schema name. In the following example, the database name is `tickit`, the schema name is `public`, and the table name is `test`.

```
create table tickit.public.test (c1 int);
```

If the database or schema doesn't exist, the table isn't created, and the statement returns an error. You can't create tables or views in the system databases `template0`, `template1`, and `padb_harvest`.

If a schema name is given, the new table is created in that schema (assuming the creator has access to the schema). The table name must be a unique name for that schema. If no schema is specified, the table is created by using the current database schema. If you are creating a temporary table, you can't specify a schema name, because temporary tables exist in a special schema.

Multiple temporary tables with the same name can exist at the same time in the same database if they are created in separate sessions because the tables are assigned to different schemas. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

#### *column\_name*

Name of a column to be created in the new table. The maximum length for the column name is 127 bytes; longer names are truncated to 127 bytes. You can use UTF-8 multibyte characters up to a maximum of four bytes. The maximum number of columns you can define in a single table is 1,600. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

##### **Note**

If you are creating a "wide table," take care that your list of columns doesn't exceed row-width boundaries for intermediate results during loads and query processing. For more information, see [Usage notes \(p. 831\)](#).

#### *data\_type*

Data type of the column being created. For CHAR and VARCHAR columns, you can use the MAX keyword instead of declaring a maximum length. MAX sets the maximum length to 4,096 bytes for CHAR or 65535 bytes for VARCHAR. The maximum size of a GEOMETRY object is 1,048,447 bytes.

For information about the data types that Amazon Redshift supports, see [Data types \(p. 574\)](#).

#### DEFAULT *default\_expr*

Clause that assigns a default data value for the column. The data type of *default\_expr* must match the data type of the column. The DEFAULT value must be a variable-free expression. Subqueries, cross-references to other columns in the current table, and user-defined functions aren't allowed.

The *default\_expr* expression is used in any INSERT operation that doesn't specify a value for the column. If no default value is specified, the default value for the column is null.

If a COPY operation with a defined column list omits a column that has a DEFAULT value, the COPY command inserts the value of *default\_expr*.

#### IDENTITY(*seed*, *step*)

Clause that specifies that the column is an IDENTITY column. An IDENTITY column contains unique autogenerated values. The data type for an IDENTITY column must be either INT or BIGINT.

When you add rows using an INSERT or INSERT INTO [tablename] VALUES( ) statement, these values start with the value specified as *seed* and increment by the number specified as *step*.

When you load the table using an INSERT INTO [tablename] SELECT \* FROM or COPY statement, the data is loaded in parallel and distributed to the node slices. To be sure that the identity values are unique, Amazon Redshift skips a number of values when creating the identity values. Identity values are unique, but the order might not match the order in the source files.

#### GENERATED BY DEFAULT AS IDENTITY(*seed*, *step*)

Clause that specifies that the column is a default IDENTITY column and enables you to automatically assign a unique value to the column. The data type for an IDENTITY column must be either INT or BIGINT. When you add rows without values, these values start with the value specified as *seed* and increment by the number specified as *step*. For information about how values are generated, see [IDENTITY \(p. 826\)](#).

Also, during INSERT, UPDATE, or COPY you can provide a value without EXPLICIT\_IDS. Amazon Redshift uses that value to insert into the identity column instead of using the system-generated value. The value can be a duplicate, a value less than the seed, or a value between step values. Amazon Redshift doesn't check the uniqueness of values in the column. Providing a value doesn't affect the next system-generated value.

#### Note

If you require uniqueness in the column, don't add a duplicate value. Instead, add a unique value that is less than the seed or between step values.

Keep in mind the following about default identity columns:

- Default identity columns are NOT NULL. NULL can't be inserted.
- To insert a generated value into a default identity column, use the keyword DEFAULT.

```
INSERT INTO tablename (identity-column-name) VALUES (DEFAULT);
```

- Overriding values of a default identity column doesn't affect the next generated value.
- You can't add a default identity column with the ALTER TABLE ADD COLUMN statement.
- You can append a default identity column with the ALTER TABLE APPEND statement.

#### ENCODE *encoding*

The compression encoding for a column. ENCODE AUTO is the default for tables. Amazon Redshift automatically manages compression encoding for all columns in the table. If you specify compression encoding for any column in the table, the table is no longer set to ENCODE AUTO.

Amazon Redshift no longer automatically manages compression encoding for all columns in the table. You can specify the ENCODE AUTO option for the table to enable Amazon Redshift to automatically manage compression encoding for all columns in the table.

Amazon Redshift automatically assigns an initial compression encoding to columns for which you don't specify compression encoding as follows:

- All columns in temporary tables are assigned RAW compression by default.
- Columns that are defined as sort keys are assigned RAW compression.
- Columns that are defined as BOOLEAN, REAL, DOUBLE PRECISION, GEOMETRY, or GEOGRAPHY data type are assigned RAW compression.
- Columns that are defined as SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP, or Timestamptz are assigned AZ64 compression.
- Columns that are defined as CHAR, VARCHAR, or VARBYTE are assigned LZO compression.

**Note**

If you don't want a column to be compressed, explicitly specify RAW encoding.

The following [compression encodings \(p. 39\)](#) are supported:

- AZ64
- BYTEDICT
- DELTA
- DELTA32K
- LZO
- MOSTLY8
- MOSTLY16
- MOSTLY32
- RAW (no compression)
- RUNLENGTH
- TEXT255
- TEXT32K
- ZSTD

**DISTKEY**

Keyword that specifies that the column is the distribution key for the table. Only one column in a table can be the distribution key. You can use the DISTKEY keyword after a column name or as part of the table definition by using the DISTKEY (*column\_name*) syntax. Either method has the same effect. For more information, see the DISTSTYLE parameter later in this topic.

The data type of a distribution key column can be: BOOLEAN, REAL, DOUBLE PRECISION, SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP, or Timestamptz, CHAR, or VARCHAR.

**SORTKEY**

Keyword that specifies that the column is the sort key for the table. When data is loaded into the table, the data is sorted by one or more columns that are designated as sort keys. You can use the SORTKEY keyword after a column name to specify a single-column sort key, or you can specify one or more columns as sort key columns for the table by using the SORTKEY (*column\_name* [, ...]) syntax. Only compound sort keys are created with this syntax.

You can define a maximum of 400 SORTKEY columns per table.

The data type of a sort key column can be: BOOLEAN, REAL, DOUBLE PRECISION, SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP, or Timestamptz, CHAR, or VARCHAR.

#### COLLATE CASE\_SENSITIVE | COLLATE CASE\_INSENSITIVE

A clause that specifies whether string search or comparison on the column is CASE\_SENSITIVE or CASE\_INSENSITIVE. The default value is the same as the current case sensitivity configuration of the database.

To find the database collation information, use the following command:

```
select db_collation();
```

```
db_collation

case_sensitive
(1 row)
```

#### NOT NULL | NULL

NOT NULL specifies that the column isn't allowed to contain null values. NULL, the default, specifies that the column accepts null values. IDENTITY columns are declared NOT NULL by default.

#### UNIQUE

Keyword that specifies that the column can contain only unique values. The behavior of the unique table constraint is the same as that for column constraints, with the additional capability to span multiple columns. To define a unique table constraint, use the UNIQUE (*column\_name* [, ... ]) syntax.

**Important**

Unique constraints are informational and aren't enforced by the system.

#### PRIMARY KEY

Keyword that specifies that the column is the primary key for the table. Only one column can be defined as the primary key by using a column definition. To define a table constraint with a multiple-column primary key, use the PRIMARY KEY (*column\_name* [, ... ]) syntax.

Identifying a column as the primary key provides metadata about the design of the schema. A primary key implies that other tables can rely on this set of columns as a unique identifier for rows. One primary key can be specified for a table, whether as a column constraint or a table constraint. The primary key constraint should name a set of columns that is different from other sets of columns named by any unique constraint defined for the same table.

**Important**

Primary key constraints are informational only. They aren't enforced by the system, but they are used by the planner.

#### References *reftable* [ ( *refcolumn* ) ]

Clause that specifies a foreign key constraint, which implies that the column must contain only values that match values in the referenced column of some row of the referenced table. The referenced columns should be the columns of a unique or primary key constraint in the referenced table.

**Important**

Foreign key constraints are informational only. They aren't enforced by the system, but they are used by the planner.

#### LIKE *parent\_table* [ { INCLUDING | EXCLUDING } DEFAULTS ]

A clause that specifies an existing table from which the new table automatically copies column names, data types, and NOT NULL constraints. The new table and the parent table are decoupled, and any changes made to the parent table aren't applied to the new table. Default expressions for

the copied column definitions are copied only if INCLUDING DEFAULTS is specified. The default behavior is to exclude default expressions, so that all columns of the new table have null defaults.

Tables created with the LIKE option don't inherit primary and foreign key constraints. Distribution style, sort keys, BACKUP, and NULL properties are inherited by LIKE tables, but you can't explicitly set them in the CREATE TABLE ... LIKE statement.

**BACKUP { YES | NO }**

A clause that specifies whether the table should be included in automated and manual cluster snapshots. For tables, such as staging tables, that don't contain critical data, specify BACKUP NO to save processing time when creating snapshots and restoring from snapshots and to reduce storage space on Amazon Simple Storage Service. The BACKUP NO setting has no affect on automatic replication of data to other nodes within the cluster, so tables with BACKUP NO specified are restored in a node failure. The default is BACKUP YES.

**DISTSTYLE { AUTO | EVEN | KEY | ALL }**

Keyword that defines the data distribution style for the whole table. Amazon Redshift distributes the rows of a table to the compute nodes according to the distribution style specified for the table. The default is AUTO.

The distribution style that you select for tables affects the overall performance of your database. For more information, see [Working with data distribution styles \(p. 49\)](#). Possible distribution styles are as follows:

- **AUTO:** Amazon Redshift assigns an optimal distribution style based on the table data. For example, if AUTO distribution style is specified, Amazon Redshift initially assigns the ALL distribution style to a small table. When the table grows larger, Amazon Redshift might change the distribution style to KEY, choosing the primary key (or a column of the composite primary key) as the DISTKEY. If the table grows larger and none of the columns are suitable to be the DISTKEY, Amazon Redshift changes the distribution style to EVEN. The change in distribution style occurs in the background with minimal impact to user queries.

To view the distribution style applied to a table, query the PG\_CLASS system catalog table. For more information, see [Viewing distribution styles \(p. 52\)](#).

- **EVEN:** The data in the table is spread evenly across the nodes in a cluster in a round-robin distribution. Row IDs are used to determine the distribution, and roughly the same number of rows are distributed to each node.
- **KEY:** The data is distributed by the values in the DISTKEY column. When you set the joining columns of joining tables as distribution keys, the joining rows from both tables are collocated on the compute nodes. When data is collocated, the optimizer can perform joins more efficiently. If you specify DISTSTYLE KEY, you must name a DISTKEY column, either for the table or as part of the column definition. For more information, see the DISTKEY parameter earlier in this topic.
- **ALL:** A copy of the entire table is distributed to every node. This distribution style ensures that all the rows required for any join are available on every node, but it multiplies storage requirements and increases the load and maintenance times for the table. ALL distribution can improve execution time when used with certain dimension tables where KEY distribution isn't appropriate, but performance improvements must be weighed against maintenance costs.

**DISTKEY ( *column\_name* )**

Constraint that specifies the column to be used as the distribution key for the table. You can use the DISTKEY keyword after a column name or as part of the table definition, by using the DISTKEY (*column\_name*) syntax. Either method has the same effect. For more information, see the DISTSTYLE parameter earlier in this topic.

**[COMPOUND | INTERLEAVED ] SORTKEY ( *column\_name* [...] ) | [ SORTKEY AUTO ]**

Specifies one or more sort keys for the table. When data is loaded into the table, the data is sorted by the columns that are designated as sort keys. You can use the SORTKEY keyword after a column

name to specify a single-column sort key, or you can specify one or more columns as sort key columns for the table by using the `SORTKEY (column_name [ , ... ] )` syntax.

You can optionally specify COMPOUND or INTERLEAVED sort style. If you specify `SORTKEY` with columns the default is COMPOUND. For more information, see [Working with sort keys \(p. 60\)](#).

If you don't specify any sort keys options, the default is AUTO.

You can define a maximum of 400 COMPOUND SORTKEY columns or 8 INTERLEAVED SORTKEY columns per table.

#### AUTO

Specifies that Amazon Redshift assigns an optimal sort key based on the table data. For example, if AUTO sort key is specified, Amazon Redshift initially assigns no sort key to a table. If Amazon Redshift determines that a sort key will improve the performance of queries, then Amazon Redshift might change the sort key of your table. The actual sorting of the table is done by automatic table sort. For more information, see [Automatic table sort \(p. 110\)](#).

Amazon Redshift doesn't modify tables that have existing sort or distribution keys. With one exception, if a table has a distribution key that has never been used in a JOIN, then the key might be changed if Amazon Redshift determines there is a better key.

To view the sort key of a table, query the `SVV_TABLE_INFO` system catalog view. For more information, see [SVV\\_TABLE\\_INFO \(p. 1457\)](#). To view the Amazon Redshift Advisor recommendations for tables, query the `SVV_ALTER_TABLE_RECOMMENDATIONS` system catalog view. For more information, see [SVV\\_ALTER\\_TABLE\\_RECOMMENDATIONS \(p. 1415\)](#). To view the actions taken by Amazon Redshift, query the `SVL_AUTO_WORKER_ACTION` system catalog view. For more information, see [SVL\\_AUTO\\_WORKER\\_ACTION \(p. 1629\)](#).

#### COMPOUND

Specifies that the data is sorted using a compound key made up of all of the listed columns, in the order they are listed. A compound sort key is most useful when a query scans rows according to the order of the sort columns. The performance benefits of sorting with a compound key decrease when queries rely on secondary sort columns. You can define a maximum of 400 COMPOUND SORTKEY columns per table.

#### INTERLEAVED

Specifies that the data is sorted using an interleaved sort key. A maximum of eight columns can be specified for an interleaved sort key.

An interleaved sort gives equal weight to each column, or subset of columns, in the sort key, so queries don't depend on the order of the columns in the sort key. When a query uses one or more secondary sort columns, interleaved sorting significantly improves query performance. Interleaved sorting carries a small overhead cost for data loading and vacuuming operations.

#### Important

Don't use an interleaved sort key on columns with monotonically increasing attributes, such as identity columns, dates, or timestamps.

#### ENCODE AUTO

Enables Amazon Redshift to automatically adjust the encoding type for all columns in the table to optimize query performance. ENCODE AUTO preserves the initial encode types that you specify in creating the table. Then, if Amazon Redshift determines that a new encoding type can improve query performance, Amazon Redshift can change the encoding type of the table columns. ENCODE AUTO is the default if you don't specify an encoding type on any column in the table.

#### UNIQUE ( *column\_name* [...] )

Constraint that specifies that a group of one or more columns of a table can contain only unique values. The behavior of the unique table constraint is the same as that for column constraints, with

the additional capability to span multiple columns. In the context of unique constraints, null values aren't considered equal. Each unique table constraint must name a set of columns that is different from the set of columns named by any other unique or primary key constraint defined for the table.

**Important**

Unique constraints are informational and aren't enforced by the system.

**PRIMARY KEY ( *column\_name* [...] )**

Constraint that specifies that a column or a number of columns of a table can contain only unique (nonduplicate) non-null values. Identifying a set of columns as the primary key also provides metadata about the design of the schema. A primary key implies that other tables can rely on this set of columns as a unique identifier for rows. One primary key can be specified for a table, whether as a single column constraint or a table constraint. The primary key constraint should name a set of columns that is different from other sets of columns named by any unique constraint defined for the same table.

**Important**

Primary key constraints are informational only. They aren't enforced by the system, but they are used by the planner.

**FOREIGN KEY ( *column\_name* [, ...] ) REFERENCES *reftable* [ ( *refcolumn* ) ]**

Constraint that specifies a foreign key constraint, which requires that a group of one or more columns of the new table must only contain values that match values in the referenced column or columns of some row of the referenced table. If *refcolumn* is omitted, the primary key of *reftable* is used. The referenced columns must be the columns of a unique or primary key constraint in the referenced table.

**Important**

Foreign key constraints are informational only. They aren't enforced by the system, but they are used by the planner.

## Usage notes

Uniqueness, primary key, and foreign key constraints are informational only; *they are not enforced by Amazon Redshift* when you populate a table. For example, if you insert data into a table with dependencies, the insert can succeed even if it violates the constraint. Nonetheless, primary keys and foreign keys are used as planning hints and they should be declared if your ETL process or some other process in your application enforces their integrity. For information about how to drop a table with dependencies, see [DROP TABLE \(p. 874\)](#).

## Limits and quotas

Consider the following limits when you create a table.

- There is a limit for the maximum number of tables in a cluster by node type. For more information, see [Limits](#) in the *Amazon Redshift Management Guide*.
- The maximum number of characters for a table name is 127.
- The maximum number of columns you can define in a single table is 1,600.
- The maximum number of SORTKEY columns you can define in a single table is 400.

## Summary of column-level settings and table-level settings

Several attributes and settings can be set at the column level or at the table level. In some cases, setting an attribute or constraint at the column level or at the table level has the same effect. In other cases, they produce different results.

The following list summarizes column-level and table-level settings:

#### DISTKEY

There is no difference in effect whether set at the column level or at the table level.

If DISTKEY is set, either at the column level or at the table level, DISTSTYLE must be set to KEY or not set at all. DISTSTYLE can be set only at the table level.

#### SORTKEY

If set at the column level, SORTKEY must be a single column. If SORTKEY is set at the table level, one or more columns can make up a compound or interleaved composite sort key.

#### COLLATE CASE\_SENSITIVE | COLLATE CASE\_INSENSITIVE

Amazon Redshift doesn't support altering case-sensitivity configuration for a column. When you append a new column to the table, Amazon Redshift uses the default value for case-sensitivity. Amazon Redshift doesn't support the COLLATE key word when appending a new column.

For information on how to create databases using database collation, see [CREATE DATABASE \(p. 745\)](#).

For information on the COLLATE function, see [COLLATE function \(p. 1285\)](#).

#### UNIQUE

At the column level, one or more keys can be set to UNIQUE; the UNIQUE constraint applies to each column individually. If UNIQUE is set at the table level, one or more columns can make up a composite UNIQUE constraint.

#### PRIMARY KEY

If set at the column level, PRIMARY KEY must be a single column. If PRIMARY KEY is set at the table level, one or more columns can make up a composite primary key .

#### FOREIGN KEY

There is no difference in effect whether FOREIGN KEY is set at the column level or at the table level. At the column level, the syntax is simply REFERENCES *reftable* [ ( *refcolumn* ) ].

## Distribution of incoming data

When the hash distribution scheme of the incoming data matches that of the target table, no physical distribution of the data is actually necessary when the data is loaded. For example, if a distribution key is set for the new table and the data is being inserted from another table that is distributed on the same key column, the data is loaded in place, using the same nodes and slices. However, if the source and target tables are both set to EVEN distribution, data is redistributed into the target table.

## Wide tables

You might be able to create a very wide table but be unable to perform query processing, such as INSERT or SELECT statements, on the table. The maximum width of a table with fixed width columns, such as CHAR, is 64KB - 1 (or 65535 bytes). If a table includes VARCHAR columns, the table can have a larger declared width without returning an error because VARCHARS columns don't contribute their full declared width to the calculated query-processing limit. The effective query-processing limit with VARCHAR columns will vary based on a number of factors.

If a table is too wide for inserting or selecting, you receive the following error.

```
ERROR: 8001
DETAIL: The combined length of columns processed in the SQL statement
```

exceeded the query-processing limit of 65535 characters (pid:7627)

## Examples

For examples that show how to use the CREATE TABLE command, see the [Examples \(p. 833\)](#) topic.

## Examples

The following examples demonstrate various column and table attributes in Amazon Redshift CREATE TABLE statements. For more information about CREATE TABLE, including parameter definitions, see [CREATE TABLE \(p. 823\)](#).

Many of the examples use tables and data from the *TICKIT* sample data set. For more information, see [Sample database](#).

You can prefix the table name with the database name and schema name in a CREATE TABLE command. For instance, `dev_database.public.sales`. The database name must be the database you are connected to. Any attempt to create database objects in another database fails with an invalid-operation error.

### Create a table with a distribution key, a compound sort key, and compression

The following example creates a SALES table in the TICKIT database with compression defined for several columns. LISTID is declared as the distribution key, and LISTID and SELLERID are declared as a multicolumn compound sort key. Primary key and foreign key constraints are also defined for the table. Prior to creating the table in the example, you might need to add a UNIQUE constraint to each column referenced by a foreign key, if constraints don't exist.

```
create table sales(
 salesid integer not null,
 listid integer not null,
 sellerid integer not null,
 buyerid integer not null,
 eventid integer not null encode mostly16,
 dateid smallint not null,
 qtysold smallint not null encode mostly8,
 pricepaid decimal(8,2) encode delta32k,
 commission decimal(8,2) encode delta32k,
 saletime timestamp,
 primary key(salesid),
 foreign key(listid) references listing(listid),
 foreign key(sellerid) references users(userid),
 foreign key(buyerid) references users(userid),
 foreign key(dateid) references date(dateid))
distkey(listid)
compound sortkey(listid,sellerid);
```

The results follow:

| schemaname | tablename | column   | type    | encoding | distkey |
|------------|-----------|----------|---------|----------|---------|
| sortkey    |           |          |         |          |         |
| public     | sales     | salesid  | integer | lzo      | false   |
| 0          | true      |          |         |          |         |
| public     | sales     | listid   | integer | none     | true    |
| 1          | true      |          |         |          |         |
| public     | sales     | sellerid | integer | none     | false   |
| 2          | true      |          |         |          |         |

|        |       |            |                             |          |       |  |
|--------|-------|------------|-----------------------------|----------|-------|--|
| public | sales | buyerid    | integer                     | lzo      | false |  |
| 0      | true  |            |                             |          |       |  |
| public | sales | eventid    | integer                     | mostly16 | false |  |
| 0      | true  |            |                             |          |       |  |
| public | sales | dateid     | smallint                    | lzo      | false |  |
| 0      | true  |            |                             |          |       |  |
| public | sales | qtysold    | smallint                    | mostly8  | false |  |
| 0      | true  |            |                             |          |       |  |
| public | sales | pricepaid  | numeric(8,2)                | delta32k | false |  |
| 0      | false |            |                             |          |       |  |
| public | sales | commission | numeric(8,2)                | delta32k | false |  |
| 0      | false |            |                             |          |       |  |
| public | sales | saletime   | timestamp without time zone | lzo      | false |  |
| 0      | false |            |                             |          |       |  |

The following example creates table t1 with a case-insensitive column col1.

```
create table T1 (
 col1 Varchar(20) collate case_insensitive
);

insert into T1 values ('bob'), ('john'), ('Tom'), ('JOHN'), ('Bob');
```

Query the table:

```
select * from T1 where col1 = 'John';

col1

john
JOHN
(2 rows)
```

## Create a table using an interleaved sort key

The following example creates the CUSTOMER table with an interleaved sort key.

```
create table customer_interleaved (
 c_custkey integer not null,
 c_name varchar(25) not null,
 c_address varchar(25) not null,
 c_city varchar(10) not null,
 c_nation varchar(15) not null,
 c_region varchar(12) not null,
 c_phone varchar(15) not null,
 c_mktsegment varchar(10) not null)
diststyle all
interleaved sortkey (c_custkey, c_city, c_mktsegment);
```

## Create a table using IF NOT EXISTS

The following example either creates the CITIES table, or does nothing and returns a message if it already exists:

```
create table if not exists cities(
 cityid integer not null,
```

```
city varchar(100) not null,
state char(2) not null);
```

## Create a table with ALL distribution

The following example creates the VENUE table with ALL distribution.

```
create table venue(
venueid smallint not null,
venuename varchar(100),
venuecity varchar(30),
venuestate char(2),
venueseats integer,
primary key(venueid))
diststyle all;
```

## Create a Table with EVEN distribution

The following example creates a table called MYEVENT with three columns.

```
create table myevent(
eventid int,
eventname varchar(200),
eventcity varchar(30))
diststyle even;
```

The table is distributed evenly and isn't sorted. The table has no declared DISTKEY or SORTKEY columns.

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'myevent';

column | type | encoding | distkey | sortkey
-----+-----+-----+-----+-----+
eventid | integer | lzo | f | 0
eventname | character varying(200) | lzo | f | 0
eventcity | character varying(30) | lzo | f | 0
(3 rows)
```

## Create a temporary table that is LIKE another table

The following example creates a temporary table called TEMPEVENT, which inherits its columns from the EVENT table.

```
create temp table tempevent(like event);
```

This table also inherits the DISTKEY and SORTKEY attributes of its parent table:

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'tempevent';

column | type | encoding | distkey | sortkey
-----+-----+-----+-----+-----+
eventid | integer | none | t | 1
venueid | smallint | none | f | 0
catid | smallint | none | f | 0
dateid | smallint | none | f | 0
eventname | character varying(200) | lzo | f | 0
starttime | timestamp without time zone | bytedict | f | 0
```

(6 rows)

## Create a table with an IDENTITY column

The following example creates a table named VENUE\_IDENT, which has an IDENTITY column named VENUEID. This column starts with 0 and increments by 1 for each record. VENUEID is also declared as the primary key of the table.

```
create table venue_ident(venueid bigint identity(0, 1),
venuename varchar(100),
venuecity varchar(30),
venuestate char(2),
venueseats integer,
primary key(venueid));
```

## Create a table with a default IDENTITY column

The following example creates a table named t1. This table has an IDENTITY column named hist\_id and a default IDENTITY column named base\_id.

```
CREATE TABLE t1(
hist_id BIGINT IDENTITY NOT NULL, /* Cannot be overridden */
base_id BIGINT GENERATED BY DEFAULT AS IDENTITY NOT NULL, /* Can be overridden */
business_key varchar(10) ,
some_field varchar(10)
);
```

Inserting a row into the table shows that both hist\_id and base\_id values are generated.

```
INSERT INTO T1 (business_key, some_field) values ('A','MM');
```

```
SELECT * FROM t1;

hist_id | base_id | business_key | some_field
-----+-----+-----+-----
1 | 1 | A | MM
```

Inserting a second row shows that the default value for base\_id is generated.

```
INSERT INTO T1 (base_id, business_key, some_field) values (DEFAULT, 'B','MNOP');
```

```
SELECT * FROM t1;

hist_id | base_id | business_key | some_field
-----+-----+-----+-----
1 | 1 | A | MM
2 | 2 | B | MNOP
```

Inserting a third row shows that the value for base\_id doesn't need to be unique.

```
INSERT INTO T1 (base_id, business_key, some_field) values (2,'B','MNNN');
```

```
SELECT * FROM t1;

hist_id | base_id | business_key | some_field
```

|   |       |      |
|---|-------|------|
| 1 | 1   A | MM   |
| 2 | 2   B | MNOP |
| 3 | 2   B | MNNN |

## Create a table with DEFAULT column values

The following example creates a CATEGORYDEF table that declares default values for each column:

```
create table categorydef(
catid smallint not null default 0,
catgroup varchar(10) default 'Special',
catname varchar(10) default 'Other',
catdesc varchar(50) default 'Special events',
primary key(catid));

insert into categorydef values(default,default,default,default);
```

```
select * from categorydef;

catid | catgroup | catname | catdesc
-----+-----+-----+-----
 0 | Special | Other | Special events
(1 row)
```

## DISTSTYLE, DISTKEY, and SORTKEY options

The following example shows how the DISTKEY, SORTKEY, and DISTSTYLE options work. In this example, COL1 is the distribution key; therefore, the distribution style must be either set to KEY or not set. By default, the table has no sort key and so isn't sorted:

```
create table t1(col1 int distkey, col2 int) diststyle key;

select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't1';

column | type | encoding | distkey | sortkey
-----+-----+-----+-----+
col1 | integer | az64 | t | 0
col2 | integer | az64 | f | 0
```

In the following example, the same column is defined as the distribution key and the sort key. Again, the distribution style must be either set to KEY or not set.

```
create table t2(col1 int distkey sortkey, col2 int);

select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't2';

column | type | encoding | distkey | sortkey
-----+-----+-----+-----+
col1 | integer | none | t | 1
col2 | integer | az64 | f | 0
```

In the following example, no column is set as the distribution key, COL2 is set as the sort key, and the distribution style is set to ALL:

```
create table t3(col1 int, col2 int sortkey) diststyle all;
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't3';

Column | Type | Encoding | DistKey | SortKey
-----+-----+-----+-----+
col1 | integer | az64 | f | 0
col2 | integer | none | f | 1
```

In the following example, the distribution style is set to EVEN and no sort key is defined explicitly; therefore the table is distributed evenly but isn't sorted.

```
create table t4(col1 int, col2 int) diststyle even;
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't4';

column | type | encoding | distkey | sortkey
-----+-----+-----+-----+
col1 | integer | az64 | f | 0
col2 | integer | az64 | f | 0
```

## Create a table with the ENCODE AUTO option

The following example creates the table t1 with automatic compression encoding. ENCODE AUTO is the default for tables when you don't specify an encoding type for any column.

```
create table t1(c0 int, c1 varchar);
```

The following example creates the table t2 with automatic compression encoding by specifying ENCODE AUTO.

```
create table t2(c0 int, c1 varchar) encode auto;
```

The following example creates the table t3 with automatic compression encoding by specifying ENCODE AUTO. Column c0 is defined with an initial encoding type of DELTA. Amazon Redshift can change the encoding if another encoding provides better query performance.

```
create table t3(c0 int encode delta, c1 varchar) encode auto;
```

The following example creates the table t4 with automatic compression encoding by specifying ENCODE AUTO. Column c0 is defined with an initial encoding of DELTA, and column c1 is defined with an initial encoding of LZO. Amazon Redshift can change these encodings if other encodings provide better query performance.

```
create table t4(c0 int encode delta, c1 varchar encode lzo) encode auto;
```

## CREATE TABLE AS

### Topics

- [Syntax \(p. 839\)](#)

- [Parameters \(p. 839\)](#)
- [CTAS usage notes \(p. 841\)](#)
- [CTAS examples \(p. 844\)](#)

Creates a new table based on a query. The owner of this table is the user that issues the command.

The new table is loaded with data defined by the query in the command. The table columns have names and data types associated with the output columns of the query. The CREATE TABLE AS (CTAS) command creates a new table and evaluates the query to load the new table.

## Syntax

```

CREATE [[LOCAL] { TEMPORARY | TEMP }]
TABLE table_name
[(column_name [, ...])]
[BACKUP { YES | NO }]
[table_attributes]
AS query

where table_attributes are:
[DISTSTYLE { AUTO | EVEN | ALL | KEY }]
[DISTKEY(distkey_identifier)]
[[COMPOUND | INTERLEAVED] SORTKEY(column_name [, ...])]

```

## Parameters

### LOCAL

Although this optional keyword is accepted in the statement, it has no effect in Amazon Redshift.  
TEMPORARY | TEMP

Creates a temporary table. A temporary table is automatically dropped at the end of the session in which it was created.

*table\_name*

The name of the table to be created.

#### Important

If you specify a table name that begins with '#', the table is created as a temporary table.  
For example:

```
create table #newtable (id) as select * from oldtable;
```

The maximum table name length is 127 bytes; longer names are truncated to 127 bytes. Amazon Redshift enforces a quota of the number of tables per cluster by node type. The table name can be qualified with the database and schema name, as the following table shows.

```
create table tickit.public.test (c1) as select * from oldtable;
```

In this example, *tickit* is the database name and *public* is the schema name. If the database or schema doesn't exist, the statement returns an error.

If a schema name is given, the new table is created in that schema (assuming the creator has access to the schema). The table name must be a unique name for that schema. If no schema is specified, the table is created using the current database schema. If you are creating a temporary table, you can't specify a schema name, since temporary tables exist in a special schema.

Multiple temporary tables with the same name are allowed to exist at the same time in the same database if they are created in separate sessions. These tables are assigned to different schemas.

*column\_name*

The name of a column in the new table. If no column names are provided, the column names are taken from the output column names of the query. Default column names are used for expressions.

BACKUP { YES | NO }

A clause that specifies whether the table should be included in automated and manual cluster snapshots. For tables, such as staging tables, that won't contain critical data, specify BACKUP NO to save processing time when creating snapshots and restoring from snapshots and to reduce storage space on Amazon Simple Storage Service. The BACKUP NO setting has no effect on automatic replication of data to other nodes within the cluster, so tables with BACKUP NO specified are restored in the event of a node failure. The default is BACKUP YES.

DISTSTYLE { AUTO | EVEN | KEY | ALL }

Defines the data distribution style for the whole table. Amazon Redshift distributes the rows of a table to the compute nodes according the distribution style specified for the table. The default is DISTSTYLE AUTO.

The distribution style that you select for tables affects the overall performance of your database. For more information, see [Working with data distribution styles \(p. 49\)](#).

- AUTO: Amazon Redshift assigns an optimal distribution style based on the table data. To view the distribution style applied to a table, query the PG\_CLASS system catalog table. For more information, see [Viewing distribution styles \(p. 52\)](#).
- EVEN: The data in the table is spread evenly across the nodes in a cluster in a round-robin distribution. Row IDs are used to determine the distribution, and roughly the same number of rows are distributed to each node. This is the default distribution method.
- KEY: The data is distributed by the values in the DISTKEY column. When you set the joining columns of joining tables as distribution keys, the joining rows from both tables are collocated on the compute nodes. When data is collocated, the optimizer can perform joins more efficiently. If you specify DISTSTYLE KEY, you must name a DISTKEY column.
- ALL: A copy of the entire table is distributed to every node. This distribution style ensures that all the rows required for any join are available on every node, but it multiplies storage requirements and increases the load and maintenance times for the table. ALL distribution can improve execution time when used with certain dimension tables where KEY distribution isn't appropriate, but performance improvements must be weighed against maintenance costs.

DISTKEY (*column*)

Specifies a column name or positional number for the distribution key. Use the name specified in either the optional column list for the table or the select list of the query. Alternatively, use a positional number, where the first column selected is 1, the second is 2, and so on. Only one column in a table can be the distribution key:

- If you declare a column as the DISTKEY column, DISTSTYLE must be set to KEY or not set at all.
- If you don't declare a DISTKEY column, you can set DISTSTYLE to EVEN.
- If you don't specify DISTKEY or DISTSTYLE, CTAS determines the distribution style for the new table based on the query plan for the SELECT clause. For more information, see [Inheritance of column and table attributes \(p. 841\)](#).

You can define the same column as the distribution key and the sort key; this approach tends to accelerate joins when the column in question is a joining column in the query.

[ COMPOUND | INTERLEAVED ] SORTKEY ( *column\_name* [ , ... ] )

Specifies one or more sort keys for the table. When data is loaded into the table, the data is sorted by the columns that are designated as sort keys.

You can optionally specify COMPOUND or INTERLEAVED sort style. The default is COMPOUND. For more information, see [Working with sort keys \(p. 60\)](#).

You can define a maximum of 400 COMPOUND SORTKEY columns or 8 INTERLEAVED SORTKEY columns per table.

If you don't specify SORTKEY, CTAS determines the sort keys for the new table based on the query plan for the SELECT clause. For more information, see [Inheritance of column and table attributes \(p. 841\)](#).

#### COMPOUND

Specifies that the data is sorted using a compound key made up of all of the listed columns, in the order they are listed. A compound sort key is most useful when a query scans rows according to the order of the sort columns. The performance benefits of sorting with a compound key decrease when queries rely on secondary sort columns. You can define a maximum of 400 COMPOUND SORTKEY columns per table.

#### INTERLEAVED

Specifies that the data is sorted using an interleaved sort key. A maximum of eight columns can be specified for an interleaved sort key.

An interleaved sort gives equal weight to each column, or subset of columns, in the sort key, so queries don't depend on the order of the columns in the sort key. When a query uses one or more secondary sort columns, interleaved sorting significantly improves query performance. Interleaved sorting carries a small overhead cost for data loading and vacuuming operations.

#### AS *query*

Any query (SELECT statement) that Amazon Redshift supports.

## CTAS usage notes

### Limits

Amazon Redshift enforces a quota of the number of tables per cluster by node type.

The maximum number of characters for a table name is 127.

The maximum number of columns you can define in a single table is 1,600.

### Inheritance of column and table attributes

CREATE TABLE AS (CTAS) tables don't inherit constraints, identity columns, default column values, or the primary key from the table that they were created from.

You can't specify column compression encodings for CTAS tables. Amazon Redshift automatically assigns compression encoding as follows:

- Columns that are defined as sort keys are assigned RAW compression.
- Columns that are defined as BOOLEAN, REAL, DOUBLE PRECISION, GEOMETRY, or GEOGRAPHY data type are assigned RAW compression.
- Columns that are defined as SMALLINT, INTEGER, BIGINT, DECIMAL, DATE, TIME, TIMETZ, TIMESTAMP, or TIMESTAMPTZ are assigned AZ64 compression.
- Columns that are defined as CHAR, VARCHAR, or VARBYTE are assigned LZO compression.

For more information, see [Compression encodings \(p. 39\)](#) and [Data types \(p. 574\)](#).

To explicitly assign column encodings, use [CREATE TABLE \(p. 823\)](#)

CTAS determines distribution style and sort key for the new table based on the query plan for the SELECT clause.

If the SELECT clause is a simple select operation from a single table, without a limit clause, order by clause, or group by clause, then CTAS uses the source table's distribution style and sort key.

For complex queries, such as queries that include joins, aggregations, an order by clause, or a limit clause, CTAS makes a best effort to choose the optimal distribution style and sort key based on the query plan.

**Note**

For best performance with large datasets or complex queries, we recommend testing using typical datasets.

You can often predict which distribution key and sort key CTAS chooses by examining the query plan to see which columns, if any, the query optimizer chooses for sorting and distributing data. If the top node of the query plan is a simple sequential scan from a single table (XN Seq Scan), then CTAS generally uses the source table's distribution style and sort key. If the top node of the query plan is anything other a sequential scan (such as XN Limit, XN Sort, XN HashAggregate, and so on), CTAS makes a best effort to choose the optimal distribution style and sort key based on the query plan.

For example, suppose you create five tables using the following types of SELECT clauses:

- A simple select statement
- A limit clause
- An order by clause using LISTID
- An order by clause using QTY SOLD
- A SUM aggregate function with a group by clause.

The following examples show the query plan for each CTAS statement.

```
explain create table sales1_simple as select listid, dateid, qtysold from sales;
 QUERY PLAN

 XN Seq Scan on sales (cost=0.00..1724.56 rows=172456 width=8)
(1 row)

explain create table sales2_limit as select listid, dateid, qtysold from sales limit 100;
 QUERY PLAN

 XN Limit (cost=0.00..1.00 rows=100 width=8)
 -> XN Seq Scan on sales (cost=0.00..1724.56 rows=172456 width=8)
(2 rows)

explain create table sales3_orderbylistid as select listid, dateid, qtysold from sales
order by listid;
 QUERY PLAN

 XN Sort (cost=1000000016724.67..1000000017155.81 rows=172456 width=8)
 Sort Key: listid
 -> XN Seq Scan on sales (cost=0.00..1724.56 rows=172456 width=8)
(3 rows)

explain create table sales4_orderbyqty as select listid, dateid, qtysold from sales order
by qtysold;
 QUERY PLAN

 XN Sort (cost=1000000016724.67..1000000017155.81 rows=172456 width=8)
 Sort Key: qtysold
```

```

-> XN Seq Scan on sales (cost=0.00..1724.56 rows=172456 width=8)
(3 rows)

explain create table sales5_groupby as select listid, dateid, sum(qtysold) from sales group
by listid, dateid;
QUERY PLAN

XN HashAggregate (cost=3017.98..3226.75 rows=83509 width=8)
-> XN Seq Scan on sales (cost=0.00..1724.56 rows=172456 width=8)
(2 rows)

```

To view the distribution key and sort key for each table, query the PG\_TABLE\_DEF system catalog table, as shown following.

```

select * from pg_table_def where tablename like 'sales%';

 tablename | column | distkey | sortkey
-----+-----+-----+-----+
 sales | salesid | f | 0
 sales | listid | t | 0
 sales | sellerid | f | 0
 sales | buyerid | f | 0
 sales | eventid | f | 0
 sales | dateid | f | 1
 sales | qtysold | f | 0
 sales | pricepaid | f | 0
 sales | commission | f | 0
 sales | saletime | f | 0
 sales1_simple | listid | t | 0
 sales1_simple | dateid | f | 1
 sales1_simple | qtysold | f | 0
 sales2_limit | listid | f | 0
 sales2_limit | dateid | f | 0
 sales2_limit | qtysold | f | 0
 sales3_orderbylistid | listid | t | 1
 sales3_orderbylistid | dateid | f | 0
 sales3_orderbylistid | qtysold | f | 0
 sales4_orderbyqty | listid | t | 0
 sales4_orderbyqty | dateid | f | 0
 sales4_orderbyqty | qtysold | f | 1
 sales5_groupby | listid | f | 0
 sales5_groupby | dateid | f | 0
 sales5_groupby | sum | f | 0

```

The following table summarizes the results. For simplicity, we omit cost, rows, and width details from the explain plan.

| Table       | CTAS SELECT statement                                     | Explain plan top node           | Dist key    | Sort key |
|-------------|-----------------------------------------------------------|---------------------------------|-------------|----------|
| S1_SIMPLE   | select listid, dateid, qtysold from sales                 | XN Seq Scan on sales ...        | LISTID      | DATEID   |
| S2_LIMIT    | select listid, dateid, qtysold from sales limit 100       | XN Limit ...                    | None (EVEN) | None     |
| S3_ORDER_BY | select listid, dateid, qtysold from sales order by listid | XN Sort ...<br>Sort Key: listid | LISTID      | LISTID   |

| Table       | CTAS SELECT statement                                                               | Explain plan top node            | Dist key    | Sort key |
|-------------|-------------------------------------------------------------------------------------|----------------------------------|-------------|----------|
| S4_ORDER_BY | <code>select listid, dateid, qtysold from sales order by qtysold</code>             | XN Sort ...<br>Sort Key: qtysold | LISTID      | QTYSOLD  |
| S5_GROUP_BY | <code>select listid, dateid, sum(qtysold) from sales group by listid, dateid</code> | XN HashAggregate ...             | None (EVEN) | None     |

You can explicitly specify distribution style and sort key in the CTAS statement. For example, the following statement creates a table using EVEN distribution and specifies SALESID as the sort key.

```
create table sales_disteven
diststyle even
sortkey (salesid)
as
select eventid, venueid, dateid, eventname
from event;
```

## Compression encoding

ENCODE AUTO is used as the default for tables. Amazon Redshift automatically manages compression encoding for all columns in the table.

## Distribution of incoming data

When the hash distribution scheme of the incoming data matches that of the target table, no physical distribution of the data is actually necessary when the data is loaded. For example, if a distribution key is set for the new table and the data is being inserted from another table that is distributed on the same key column, the data is loaded in place, using the same nodes and slices. However, if the source and target tables are both set to EVEN distribution, data is redistributed into the target table.

## Automatic ANALYZE operations

Amazon Redshift automatically analyzes tables that you create with CTAS commands. You do not need to run the ANALYZE command on these tables when they are first created. If you modify them, you should analyze them in the same way as other tables.

## CTAS examples

The following example creates a table called EVENT\_BACKUP for the EVENT table:

```
create table event_backup as select * from event;
```

The resulting table inherits the distribution and sort keys from the EVENT table.

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'event_backup';

column | type | encoding | distkey | sortkey
-----+-----+-----+-----+-----+
catid | smallint | none | false | 0
dateid | smallint | none | false | 1
eventid | integer | none | true | 0
```

|           |                             |      |       |   |
|-----------|-----------------------------|------|-------|---|
| eventname | character varying(200)      | none | false | 0 |
| starttime | timestamp without time zone | none | false | 0 |
| venueid   | smallint                    | none | false | 0 |

The following command creates a new table called EVENTDISTSORT by selecting four columns from the EVENT table. The new table is distributed by EVENTID and sorted by EVENTID and DATEID:

```
create table eventdistsort
distkey (1)
sortkey (1,3)
as
select eventid, venueid, dateid, eventname
from event;
```

The result is as follows:

|           |                        |          |         |         |
|-----------|------------------------|----------|---------|---------|
| column    | type                   | encoding | distkey | sortkey |
| eventid   | integer                | none     | t       | 1       |
| venueid   | smallint               | none     | f       | 0       |
| dateid    | smallint               | none     | f       | 2       |
| eventname | character varying(200) | none     | f       | 0       |

You could create exactly the same table by using column names for the distribution and sort keys. For example:

```
create table eventdistsort1
distkey (eventid)
sortkey (eventid, dateid)
as
select eventid, venueid, dateid, eventname
from event;
```

The following statement applies even distribution to the table but doesn't define an explicit sort key.

```
create table eventdisteven
diststyle even
as
select eventid, venueid, dateid, eventname
from event;
```

The table doesn't inherit the sort key from the EVENT table (EVENTID) because EVEN distribution is specified for the new table. The new table has no sort key and no distribution key.

|           |                        |          |         |         |
|-----------|------------------------|----------|---------|---------|
| column    | type                   | encoding | distkey | sortkey |
| eventid   | integer                | none     | f       | 0       |
| venueid   | smallint               | none     | f       | 0       |
| dateid    | smallint               | none     | f       | 0       |
| eventname | character varying(200) | none     | f       | 0       |

The following statement applies even distribution and defines a sort key:

```
create table eventdisteventsort diststyle even sortkey (venueid)
as select eventid, venueid, dateid, eventname from event;
```

The resulting table has a sort key but no distribution key.

| column    | type                   | encoding | distkey | sortkey |
|-----------|------------------------|----------|---------|---------|
| eventid   | integer                | none     | f       | 0       |
| venueid   | smallint               | none     | f       | 1       |
| dateid    | smallint               | none     | f       | 0       |
| eventname | character varying(200) | none     | f       | 0       |

The following statement redistributes the EVENT table on a different key column from the incoming data, which is sorted on the EVENTID column, and defines no SORTKEY column; therefore the table isn't sorted.

```
create table venuedistevent distkey(venueid)
as select * from event;
```

The result is as follows:

| column    | type                        | encoding | distkey | sortkey |
|-----------|-----------------------------|----------|---------|---------|
| eventid   | integer                     | none     | f       | 0       |
| venueid   | smallint                    | none     | t       | 0       |
| catid     | smallint                    | none     | f       | 0       |
| dateid    | smallint                    | none     | f       | 0       |
| eventname | character varying(200)      | none     | f       | 0       |
| starttime | timestamp without time zone | none     | f       | 0       |

## CREATE USER

Creates a new database user. Database users can retrieve data, run commands, and perform other actions in a database, depending on their privileges and roles. You must be a database superuser to run this command.

### Required privileges

Following are required privileges for CREATE USER:

- Superuser
- Users with the CREATE USER privilege

### Syntax

```
CREATE USER name [WITH]
PASSWORD { 'password' | 'md5hash' | 'sha256hash' | DISABLE }
[option [...]]
```

where *option* can be:

```
CREATEDB | NOCREATEDB
| CREATEUSER | NOCREATEUSER
| SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }
| IN GROUP groupname [, ...]
| VALID UNTIL 'abstime'
| CONNECTION LIMIT { limit | UNLIMITED }
| SESSION TIMEOUT limit
| EXTERNALID external_id
```

## Parameters

*name*

The name of the user to create. The user name can't be PUBLIC. For more information about valid names, see [Names and identifiers \(p. 571\)](#).

WITH

Optional keyword. WITH is ignored by Amazon Redshift

```
PASSWORD { 'password' | 'md5hash' | 'sha256hash' | DISABLE }
```

Sets the user's password.

By default, users can change their own passwords, unless the password is disabled. To disable a user's password, specify DISABLE. When a user's password is disabled, the password is deleted from the system and the user can log on only using temporary AWS Identity and Access Management (IAM) user credentials. For more information, see [Using IAM Authentication to Generate Database User Credentials](#). Only a superuser can enable or disable passwords. You can't disable a superuser's password. To enable a password, run [ALTER USER \(p. 659\)](#) and specify a password.

You can specify the password in clear text, as an MD5 hash string, or as a SHA256 hash string.

**Note**

When you launch a new cluster using the AWS Management Console, AWS CLI, or Amazon Redshift API, you must supply a clear text password for the initial database user. You can change the password later by using [ALTER USER \(p. 659\)](#).

For clear text, the password must meet the following constraints:

- It must be 8 to 64 characters in length.
- It must contain at least one uppercase letter, one lowercase letter, and one number.
- It can use any ASCII characters with ASCII codes 33–126, except ' (single quotation mark), " (double quotation mark), \, /, or @.

As a more secure alternative to passing the CREATE USER password parameter as clear text, you can specify an MD5 hash of a string that includes the password and user name.

**Note**

When you specify an MD5 hash string, the CREATE USER command checks for a valid MD5 hash string, but it doesn't validate the password portion of the string. It is possible in this case to create a password, such as an empty string, that you can't use to log on to the database.

To specify an MD5 password, follow these steps:

1. Concatenate the password and user name.

For example, for password ez and user user1, the concatenated string is ezuser1.

2. Convert the concatenated string into a 32-character MD5 hash string. You can use any MD5 utility to create the hash string. The following example uses the Amazon Redshift [MD5 function \(p. 1134\)](#) and the concatenation operator ( || ) to return a 32-character MD5-hash string.

```
select md5('ez' || 'user1');

md5

153c434b4b77c89e6b94f12c5393af5b
```

3. Concatenate 'md5' in front of the MD5 hash string and provide the concatenated string as the *md5hash* argument.

```
create user user1 password 'md5153c434b4b77c89e6b94f12c5393af5b';
```

4. Log on to the database using the sign-in credentials.

For this example, log on as *user1* with password *ez*.

Another secure alternative is to specify an SHA-256 hash of a password string; or you can provide your own valid SHA-256 digest and 256-bit salt that was used to create the digest.

- Digest – The output of a hashing function.
- Salt – Randomly generated data that is combined with the password to help reduce patterns in the hashing function output.

```
'sha256|Mypassword'
```

```
'sha256|digest|256-bit-salt'
```

In the following example, Amazon Redshift generates and manages the salt.

```
CREATE USER admin PASSWORD 'sha256|Mypassword1';
```

In the following example, a valid SHA-256 digest and 256-bit salt that was used to create the digest are supplied.

To specify a password and hash it with your own salt, follow these steps:

1. Create a 256-bit salt. You can obtain a salt by using any hexadecimal string generator to generate a string 64 characters long. For this example, the salt is *c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6*.
2. Use the FROM\_HEX function to convert your salt to binary. This is because the SHA2 function requires the binary representation of the salt. See the following statement.

```
SELECT FROM_HEX('c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6');
```

3. Use the CONCAT function to append your salt to your password. For this example, the password is *Mypassword1*. See the following statement.

```
SELECT
CONCAT('Mypassword1',FROM_HEX('c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6'))
```

4. Use the SHA2 function to create a digest from your password and salt combination. See the following statement.

```
SELECT
SHA2(CONCAT('Mypassword1', FROM_HEX('c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6'));
```

5. Using the digest and salt from the previous steps, create the user. See the following statement.

```
CREATE USER admin PASSWORD 'sha256|
821708135fcc42eb3afda85286dee0ed15c2c461d000291609f77eb113073ec2|
c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6';
```

6. Log on to the database using the sign-in credentials.

For this example, log on as admin with password Mypassword1.

If you set a password in plain text without specifying the hashing function, then an MD5 digest is generated using the username as the salt.

**CREATEDB | NOCREATEDB**

The CREATEDB option allows the new user to create databases. The default is NOCREATEDB.

**CREATEUSER | NOCREATEUSER**

The CREATEUSER option creates a superuser with all database privileges, including CREATE USER. The default is NOCREATEUSER. For more information, see [superuser \(p. 537\)](#).

**SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }**

A clause that specifies the level of access the user has to the Amazon Redshift system tables and views.

If RESTRICTED is specified, the user can see only the rows generated by that user in user-visible system tables and views. The default is RESTRICTED.

If UNRESTRICTED is specified, the user can see all rows in user-visible system tables and views, including rows generated by another user. UNRESTRICTED doesn't give a regular user access to superuser-visible tables. Only superusers can see superuser-visible tables.

**Note**

Giving a user unrestricted access to system tables gives the user visibility to data generated by other users. For example, STL\_QUERY and STL\_QUERYTEXT contain the full text of INSERT, UPDATE, and DELETE statements, which might contain sensitive user-generated data.

All rows in SVV\_TRANSACTIONS are visible to all users.

For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

**IN GROUP *groupname***

Specifies the name of an existing group that the user belongs to. Multiple group names may be listed.

**VALID UNTIL *abstime***

The VALID UNTIL option sets an absolute time after which the user's password is no longer valid. By default the password has no time limit.

**CONNECTION LIMIT { *limit* | UNLIMITED }**

The maximum number of database connections the user is permitted to have open concurrently. The limit isn't enforced for superusers. Use the UNLIMITED keyword to permit the maximum number of

concurrent connections. A limit on the number of connections for each database might also apply. For more information, see [CREATE DATABASE \(p. 745\)](#). The default is UNLIMITED. To view current connections, query the [STV\\_SESSIONS \(p. 1597\)](#) system view.

**Note**

If both user and database connection limits apply, an unused connection slot must be available that is within both limits when a user attempts to connect.

**SESSION TIMEOUT *limit***

The maximum time in seconds that a session remains inactive or idle. The range is 60 seconds (one minute) to 1,728,000 seconds (20 days). If no session timeout is set for the user, the cluster setting applies. For more information, see [Quotas and limits in Amazon Redshift](#) in the *Amazon Redshift Management Guide*.

When you set the session timeout, it's applied to new sessions only.

To view information about active user sessions, including the start time, user name, and session timeout, query the [STV\\_SESSIONS \(p. 1597\)](#) system view. To view information about user-session history, query the [STL\\_SESSIONS \(p. 1557\)](#) view. To retrieve information about database users, including session-timeout values, query the [SVL\\_USER\\_INFO \(p. 1669\)](#) view.

**EXTERNALID *external\_id***

The identifier for the user, which is associated with an identity provider. The user must have their password disabled. For more information, see [Native identity provider \(IdP\) federation for Amazon Redshift](#).

## Usage notes

By default, all users have CREATE and USAGE privileges on the PUBLIC schema. To disallow users from creating objects in the PUBLIC schema of a database, use the REVOKE command to remove that privilege.

When using IAM authentication to create database user credentials, you might want to create a superuser that is able to log on only using temporary credentials. You can't disable a superuser's password, but you can create an unknown password using a randomly generated MD5 hash string.

```
create user iam_superuser password 'md5A1234567890123456780123456789012' createuser;
```

The case of a *username* enclosed in double quotation marks is always preserved regardless of the setting of the `enable_case_sensitive_identifier` configuration option. For more information, see [enable\\_case\\_sensitive\\_identifier \(p. 1688\)](#).

## Examples

The following command creates a user named dbuser, with the password "abcD1234", database creation privileges, and a connection limit of 30.

```
create user dbuser with password 'abcD1234' createdb connection limit 30;
```

Query the PG\_USER\_INFO catalog table to view details about a database user.

```
select * from pg_user_info;

username | usesysid | usecreatedb | usesuper | usecatupd | passwd | valuntil |
useconfig | useconnlimit
```

| rdsdb     |  | 1   | true | true  | true  | ***** | infinity |  |
|-----------|--|-----|------|-------|-------|-------|----------|--|
|           |  |     |      |       |       |       |          |  |
| adminuser |  | 100 | true | true  | false | ***** |          |  |
| UNLIMITED |  |     |      |       |       |       |          |  |
| dbuser    |  | 102 | true | false | false | ***** |          |  |
| 30        |  |     |      |       |       |       |          |  |

In the following example, the account password is valid until June 10, 2017.

```
create user dbuser with password 'abcD1234' valid until '2017-06-10';
```

The following example creates a user with a case-sensitive password that contains special characters.

```
create user newman with password '@AbC4321!';
```

To use a backslash ('\') in your MD5 password, escape the backslash with a backslash in your source string. The following example creates a user named slashpass with a single backslash ('\\') as the password.

```
select md5('\\\\'||'slashpass');
md5

0c983d1a624280812631c5389e60d48c
```

Create a user with the md5 password.

```
create user slashpass password 'md50c983d1a624280812631c5389e60d48c';
```

The following example creates a user named dbuser with an idle-session timeout set to 120 seconds.

```
CREATE USER dbuser password 'abcD1234' SESSION TIMEOUT 120;
```

The following example creates a user named bob. The namespace is myco\_aad. This is only a sample. To run the command successfully, you must have a registered identity provider. For more information, see [Native identity provider \(IdP\) federation for Amazon Redshift](#).

```
CREATE USER myco_aad:bob EXTERNALID "ABC123" PASSWORD DISABLE;
```

## CREATE VIEW

Creates a view in a database. The view isn't physically materialized; the query that defines the view is run every time the view is referenced in a query. To create a view with an external table, include the WITH NO SCHEMA BINDING clause.

To create a standard view, you need access to the underlying tables, or to underlying views. To query a standard view, you need select privileges for the view itself, but you don't need select privileges for the underlying tables. To query a late binding view, you need select privileges for the late binding view itself. You should also make sure the owner of the late binding view has select privileges to the referenced objects (tables, views, or user-defined functions). For more information about Late Binding Views, see [Usage notes \(p. 853\)](#).

## Required privileges

Following are required privileges for CREATE VIEW:

- For CREATE VIEW:
  - Superuser
  - Users with the CREATE [ OR REPLACE ] VIEW privilege
- For REPLACE VIEW:
  - Superuser
  - Users with the CREATE [ OR REPLACE ] VIEW privilege
  - View owner

## Syntax

```
CREATE [OR REPLACE] VIEW name [(column_name [, ...])] AS query
[WITH NO SCHEMA BINDING]
```

## Parameters

### OR REPLACE

If a view of the same name already exists, the view is replaced. You can only replace a view with a new query that generates the identical set of columns, using the same column names and data types. CREATE OR REPLACE VIEW locks the view for reads and writes until the operation completes.

When a view is replaced, its other properties such as ownership and granted privileges are preserved.

### *name*

The name of the view. If a schema name is given (such as myschema.myview) the view is created using the specified schema. Otherwise, the view is created in the current schema. The view name must be different from the name of any other view or table in the same schema.

If you specify a view name that begins with '#', the view is created as a temporary view that is visible only in the current session.

For more information about valid names, see [Names and identifiers \(p. 571\)](#). You can't create tables or views in the system databases template0, template1, and padb\_harvest.

### *column\_name*

Optional list of names to be used for the columns in the view. If no column names are given, the column names are derived from the query. The maximum number of columns you can define in a single view is 1,600.

### *query*

A query (in the form of a SELECT statement) that evaluates to a table. This table defines the columns and rows in the view.

### WITH NO SCHEMA BINDING

Clause that specifies that the view isn't bound to the underlying database objects, such as tables and user-defined functions. As a result, there is no dependency between the view and the objects it references. You can create a view even if the referenced objects don't exist. Because there is no dependency, you can drop or alter a referenced object without affecting the view. Amazon Redshift doesn't check for dependencies until the view is queried. To view details about late binding views, run the [PG\\_GET\\_LATE\\_BINDING\\_VIEW\\_COLS \(p. 1398\)](#) function.

When you include the WITH NO SCHEMA BINDING clause, tables and views referenced in the SELECT statement must be qualified with a schema name. The schema must exist when the view is created, even if the referenced table doesn't exist. For example, the following statement returns an error.

```
create view myevent as select eventname from event
with no schema binding;
```

The following statement runs successfully.

```
create view myevent as select eventname from public.event
with no schema binding;
```

#### **Note**

You can't update, insert into, or delete from a view.

## Usage notes

### Late-binding views

A late-binding view doesn't check the underlying database objects, such as tables and other views, until the view is queried. As a result, you can alter or drop the underlying objects without dropping and recreating the view. If you drop underlying objects, queries to the late-binding view will fail. If the query to the late-binding view references columns in the underlying object that aren't present, the query will fail.

If you drop and then re-create a late-binding view's underlying table or view, the new object is created with default access permissions. You might need to grant permissions to the underlying objects for users who will query the view.

To create a late-binding view, include the WITH NO SCHEMA BINDING clause. The following example creates a view with no schema binding.

```
create view event_vw as select * from public.event
with no schema binding;
```

```
select * from event_vw limit 1;

eventid | venueid | catid | dateid | eventname | starttime
-----+-----+-----+-----+-----+-----+
 2 | 306 | 8 | 2114 | Boris Godunov | 2008-10-15 20:00:00
```

The following example shows that you can alter an underlying table without recreating the view.

```
alter table event rename column eventname to title;
```

```
select * from event_vw limit 1;

eventid | venueid | catid | dateid | title | starttime
-----+-----+-----+-----+-----+-----+
 2 | 306 | 8 | 2114 | Boris Godunov | 2008-10-15 20:00:00
```

You can reference Amazon Redshift Spectrum external tables only in a late-binding view. One application of late-binding views is to query both Amazon Redshift and Redshift Spectrum tables. For example, you

can use the [UNLOAD \(p. 991\)](#) command to archive older data to Amazon S3. Then, create a Redshift Spectrum external table that references the data on Amazon S3 and create a view that queries both tables. The following example uses a UNION ALL clause to join the Amazon Redshift SALES table and the Redshift Spectrum SPECTRUM.SALES table.

```
create view sales_vw as
select * from public.sales
union all
select * from spectrum.sales
with no schema binding;
```

For more information about creating Redshift Spectrum external tables, including the SPECTRUM.SALES table, see [Getting started with Amazon Redshift Spectrum \(p. 250\)](#).

## Examples

The example commands use a sample set of objects and data called the *TICKIT* database. For more information, see [Sample database](#).

The following command creates a view called *myevent* from a table called EVENT.

```
create view myevent as select eventname from event
where eventname = 'LeAnn Rimes';
```

The following command creates a view called *myuser* from a table called USERS.

```
create view myuser as select lastname from users;
```

The following command creates or replaces a view called *myuser* from a table called USERS.

```
create or replace view myuser as select lastname from users;
```

The following example creates a view with no schema binding.

```
create view myevent as select eventname from public.event
with no schema binding;
```

## DEALLOCATE

Deallocates a prepared statement.

### Syntax

```
DEALLOCATE [PREPARE] plan_name
```

### Parameters

PREPARE

This keyword is optional and is ignored.

*plan\_name*

The name of the prepared statement to deallocate.

## Usage Notes

DEALLOCATE is used to deallocate a previously prepared SQL statement. If you don't explicitly deallocate a prepared statement, it is deallocated when the current session ends. For more information on prepared statements, see [PREPARE \(p. 912\)](#).

## See Also

[EXECUTE \(p. 880\)](#), [PREPARE \(p. 912\)](#)

## DECLARE

Defines a new cursor. Use a cursor to retrieve a few rows at a time from the result set of a larger query.

When the first row of a cursor is fetched, the entire result set is materialized on the leader node, in memory or on disk, if needed. Because of the potential negative performance impact of using cursors with large result sets, we recommend using alternative approaches whenever possible. For more information, see [Performance considerations when using cursors \(p. 856\)](#).

You must declare a cursor within a transaction block. Only one cursor at a time can be open per session.

For more information, see [FETCH \(p. 886\)](#), [CLOSE \(p. 674\)](#).

## Syntax

```
DECLARE cursor_name CURSOR FOR query
```

## Parameters

*cursor\_name*

Name of the new cursor.

*query*

A SELECT statement that populates the cursor.

## DECLARE CURSOR usage notes

If your client application uses an ODBC connection and your query creates a result set that is too large to fit in memory, you can stream the result set to your client application by using a cursor. When you use a cursor, the entire result set is materialized on the leader node, and then your client can fetch the results incrementally.

### Note

To enable cursors in ODBC for Microsoft Windows, enable the **Use Declare/Fetch** option in the ODBC DSN you use for Amazon Redshift. We recommend setting the ODBC cache size, using the **Cache Size** field in the ODBC DSN options dialog, to 4,000 or greater on multi-node clusters to minimize round trips. On a single-node cluster, set Cache Size to 1,000.

Because of the potential negative performance impact of using cursors, we recommend using alternative approaches whenever possible. For more information, see [Performance considerations when using cursors \(p. 856\)](#).

Amazon Redshift cursors are supported with the following limitations:

- Only one cursor at a time can be open per session.

- Cursors must be used within a transaction (BEGIN ... END).
- The maximum cumulative result set size for all cursors is constrained based on the cluster node type. If you need larger result sets, you can resize to an XL or 8XL node configuration.

For more information, see [Cursor constraints \(p. 856\)](#).

## Cursor constraints

When the first row of a cursor is fetched, the entire result set is materialized on the leader node. If the result set doesn't fit in memory, it is written to disk as needed. To protect the integrity of the leader node, Amazon Redshift enforces constraints on the size of all cursor result sets, based on the cluster's node type.

The following table shows the maximum total result set size for each cluster node type. Maximum result set sizes are in megabytes.

| Node type                     | Maximum result set per cluster (MB) |
|-------------------------------|-------------------------------------|
| DS1 or DS2 XL single node     | 64000                               |
| DS1 or DS2 XL multiple nodes  | 1800000                             |
| DS1 or DS2 8XL multiple nodes | 14400000                            |
| RA3 16XL multiple nodes       | 14400000                            |
| DC1 Large single node         | 16000                               |
| DC1 Large multiple nodes      | 384000                              |
| DC1 8XL multiple nodes        | 3000000                             |
| DC2 Large single node         | 8000                                |
| DC2 Large multiple nodes      | 192000                              |
| DC2 8XL multiple nodes        | 3200000                             |
| RA3 4XL multiple nodes        | 3200000                             |
| RA3 XLPLUS multiple nodes     | 1000000                             |
| RA3 XLPLUS single node        | 64000                               |
| Amazon Redshift Serverless    | 150000                              |

To view the active cursor configuration for a cluster, query the [STV\\_CURSOR\\_CONFIGURATION \(p. 1581\)](#) system table as a superuser. To view the state of active cursors, query the [STV\\_ACTIVE\\_CURSORS \(p. 1578\)](#) system table. Only the rows for a user's own cursors are visible to the user, but a superuser can view all cursors.

## Performance considerations when using cursors

Because cursors materialize the entire result set on the leader node before beginning to return results to the client, using cursors with very large result sets can have a negative impact on performance. We strongly recommend against using cursors with very large result sets. In some cases, such as when your application uses an ODBC connection, cursors might be the only feasible solution. If possible, we recommend using these alternatives:

- Use [UNLOAD \(p. 991\)](#) to export a large table. When you use UNLOAD, the compute nodes work in parallel to transfer the data directly to data files on Amazon Simple Storage Service. For more information, see [Unloading data \(p. 144\)](#).
- Set the JDBC fetch size parameter in your client application. If you use a JDBC connection and you are encountering client-side out-of-memory errors, you can enable your client to retrieve result sets in smaller batches by setting the JDBC fetch size parameter. For more information, see [Setting the JDBC fetch size parameter \(p. 489\)](#).

## DECLARE CURSOR examples

The following example declares a cursor named LOLLAPALOOZA to select sales information for the Lollapalooza event, and then fetches rows from the result set using the cursor:

```
-- Begin a transaction
begin;

-- Declare a cursor
declare lollapalooza cursor for
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Lollapalooza';

-- Fetch the first 5 rows in the cursor lollapalooza:
fetch forward 5 from lollapalooza;

eventname | starttime | costperticket | qtysold
-----+-----+-----+-----+
Lollapalooza | 2008-05-01 19:00:00 | 92.00000000 | 3
Lollapalooza | 2008-11-15 15:00:00 | 222.00000000 | 2
Lollapalooza | 2008-04-17 15:00:00 | 239.00000000 | 3
Lollapalooza | 2008-04-17 15:00:00 | 239.00000000 | 4
Lollapalooza | 2008-04-17 15:00:00 | 239.00000000 | 1
(5 rows)

-- Fetch the next row:
fetch next from lollapalooza;

eventname | starttime | costperticket | qtysold
-----+-----+-----+-----+
Lollapalooza | 2008-10-06 14:00:00 | 114.00000000 | 2

-- Close the cursor and end the transaction:
close lollapalooza;
commit;
```

The following example loops over a refcursor with all the results from a table:

```
CREATE TABLE tbl_1 (a int, b int);
INSERT INTO tbl_1 values (1, 2),(3, 4);

CREATE OR REPLACE PROCEDURE sp_cursor_loop() AS $$
DECLARE
 target record;
 curs1 cursor for select * from tbl_1;
BEGIN
```

```

OPEN curs1;
LOOP
 fetch curs1 into target;
 exit when not found;
 RAISE INFO 'a %', target.a;
END LOOP;
CLOSE curs1;
END;
$$ LANGUAGE plpgsql;

CALL sp_cursor_loop();

SELECT message
 from svl_stored_proc_messages
 where querytxt like 'CALL sp_cursor_loop()%';

message

 a 1
 a 3

```

## DELETE

Deletes rows from tables.

**Note**

The maximum size for a single SQL statement is 16 MB.

## Syntax

```

[WITH [RECURSIVE] common_table_expression [, common_table_expression , ...]]
DELETE [FROM] table_name
 [{USING} table_name, ...]
 [WHERE condition]

```

## Parameters

### WITH clause

Optional clause that specifies one or more *common-table-expressions*. See [WITH clause \(p. 929\)](#).

### FROM

The FROM keyword is optional, except when the USING clause is specified. The statements delete from event; and delete event; are equivalent operations that remove all of the rows from the EVENT table.

**Note**

To delete all the rows from a table, [TRUNCATE \(p. 990\)](#) the table. TRUNCATE is much more efficient than DELETE and doesn't require a VACUUM and ANALYZE. However, be aware that TRUNCATE commits the transaction in which it is run.

### *table\_name*

A temporary or persistent table. Only the owner of the table or a user with DELETE privilege on the table may delete rows from the table.

Consider using the TRUNCATE command for fast unqualified delete operations on large tables; see [TRUNCATE \(p. 990\)](#).

**Note**

After deleting a large number of rows from a table:

- Vacuum the table to reclaim storage space and re-sort rows.
- Analyze the table to update statistics for the query planner.

**USING** *table\_name*, ...

The **USING** keyword is used to introduce a table list when additional tables are referenced in the **WHERE** clause condition. For example, the following statement deletes all of the rows from the **EVENT** table that satisfy the join condition over the **EVENT** and **SALES** tables. The **SALES** table must be explicitly named in the **FROM** list:

```
delete from event using sales where event.eventid=sales.eventid;
```

If you repeat the target table name in the **USING** clause, the **DELETE** operation runs a self-join. You can use a subquery in the **WHERE** clause instead of the **USING** syntax as an alternative way to write the same query.

**WHERE** *condition*

Optional clause that limits the deletion of rows to those that match the condition. For example, the condition can be a restriction on a column, a join condition, or a condition based on the result of a query. The query can reference tables other than the target of the **DELETE** command. For example:

```
delete from t1
where col1 in(select col2 from t2);
```

If no condition is specified, all of the rows in the table are deleted.

## Examples

Delete all of the rows from the **CATEGORY** table:

```
delete from category;
```

Delete rows with **CATID** values between 0 and 9 from the **CATEGORY** table:

```
delete from category
where catid between 0 and 9;
```

Delete rows from the **LISTING** table whose **SELLERID** values don't exist in the **SALES** table:

```
delete from listing
where listing.sellerid not in(select sales.sellerid from sales);
```

The following two queries both delete one row from the **CATEGORY** table, based on a join to the **EVENT** table and an additional restriction on the **CATID** column:

```
delete from category
using event
where event.catid=category.catid and category.catid=9;
```

```
delete from category
where catid in
(select category.catid from category, event
where category.catid=event.catid and category.catid=9);
```

## DESC DATASHARE

Displays a list of the database objects within a datashare that are added to it using ALTER DATASHARE. Amazon Redshift displays the names, databases, schemas, and types of tables, views, and functions.

### Syntax

```
DESC DATASHARE datashare_name [OF [ACCOUNT account_id] NAMESPACE namespace_guid]
```

### Parameters

*datashare\_name*

The name of the datashare .

NAMESPACE *namespace\_guid*

A value that specifies the namespace that the datashare uses. When you run DESC DATAHSARE as a consumer cluster administrator, specify the NAMESPACE parameter to view inbound datashares.

ACCOUNT *account\_id*

A value that specifies the account that the datashare belongs to.

### Usage Notes

As a consumer account administrator, when you run DESC DATASHARE to see inbound datashares within the AWS account, specify the NAMESPACE option. When you run DESC DATASHARE to see inbound datashares across AWS accounts, specify the ACCOUNT and NAMESPACE options.

### Examples

The following example displays the information for outbound datashares on a producer cluster.

```
DESC DATASHARE salesshare;

producer_account | producer_namespace | share_type | share_name |
object_type | object_name | include_new
-----+-----+-----+-----+
+-----+-----+-----+-----+
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salesshare |
TABLE | public.ticket_sales_redshift |
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND | salesshare |
SCHEMA | public | t
```

The following example displays the information for inbound datashares on a consumer cluster.

```
DESC DATASHARE salesshare of ACCOUNT '123456789012' NAMESPACE
'13b8833d-17c6-4f16-8fe4-1a018f5ed00d';

producer_account | producer_namespace | share_type | share_name |
object_type | object_name | include_new
-----+-----+-----+-----+
+-----+-----+-----+-----+
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare | table
| public.ticket_sales_redshift |
```

```
123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND | salesshare | schema
| public | |
(2 rows)
```

## DESC IDENTITY PROVIDER

Displays information about an identity provider. Only a superuser can describe an identity provider.

## Syntax

DESC IDENTITY PROVIDER *identity\_provider\_name*

## Parameters

*identity\_provider\_name*

The name of the identity provider.

## Example

The following example displays information about the identity provider.

DESC IDENTITY PROVIDER azure\_idp;

## Sample output.

```
uid | name | type | instanceid | namespc |
 | | | | |
 params
 | enabled
-----+-----+-----+-----+
+-----+
+-----+
126692 | azure_idp | azure | e40d4bb2-7670-44ae-bfb8-5db013221d73 | aad | {"issuer":"https://login.microsoftonline.com/e40d4bb2-7670-44ae-bfb8-5db013221d73/v2.0", "client_id":"871c010f-5e61-4fb1-83ac-98610a7e9110", "client_secret":'', "audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift", "https://analysis.windows.net/powerbi/connector/AWSRDS"]}] | t
(1 row)
```

## DETACH MASKING POLICY

Detaches an already attached dynamic data masking policy from a column. For more information on dynamic data masking, see [Dynamic data masking \(p. 555\)](#).

Superusers and users or roles that have the sys:secadmin role can detach a masking policy.

## Syntax

```
DETACH MASKING POLICY policy_name
 ON { table_name }
 (output_column_names)
```

```
FROM { user_name | ROLE role_name | PUBLIC };
```

## Parameters

*policy\_name*

The name of the masking policy to detach.

*table\_name*

The name of the table to detach the masking policy from.

*output\_column\_names*

The names of the columns to which the masking policy was attached.

*user\_name*

The name of the user to whom the masking policy was attached.

You can only set one of *user\_name*, *role\_name*, and PUBLIC in a single DETACH MASKING POLICY statement.

*role\_name*

The name of the role to which the masking policy was attached.

You can only set one of *user\_name*, *role\_name*, and PUBLIC in a single DETACH MASKING POLICY statement.

*PUBLIC*

Shows that the policy was attached to all users in the table.

You can only set one of *user\_name*, *role\_name*, and PUBLIC in a single DETACH MASKING POLICY statement.

## DETACH RLS POLICY

Detach a row-level security policy on a table from one or more users or roles.

Superusers and users or roles that have the sys:secadmin role can detach a policy.

## Syntax

```
DETACH RLS POLICY policy_name ON [TABLE] table_name [, ...]
FROM { user_name | ROLE role_name | PUBLIC } [, ...]
```

## Parameters

*policy\_name*

The name of the policy.

ON [TABLE] *table\_name* [, ...]

The table or view that the row-level security policy is detached from.

FROM { *user\_name* | ROLE *role\_name* | PUBLIC } [, ...]

Specifies whether the policy is detached from one or more specified users or roles.

## Usage notes

When working with the DETACH RLS POLICY statement, observe the following:

- You can detach a policy from a relation, user, role, or public.

## Examples

The following example detaches a policy on a table from a role.

```
DETACH RLS POLICY policy_concerts ON tickit_category_redshift FROM ROLE analyst, ROLE dbadmin;
```

## DROP DATABASE

Drops a database.

You can't run DROP DATABASE within a transaction block (BEGIN ... END). For more information about transactions, see [Serializable isolation \(p. 119\)](#).

## Syntax

```
DROP DATABASE database_name
```

## Parameters

*database\_name*

Name of the database to be dropped. You can't drop the dev, padb\_harvest, template0, or template1 databases, and you can't drop the current database.

To drop an external database, drop the external schema. For more information, see [DROP SCHEMA \(p. 872\)](#).

## DROP DATABASE usage notes

When using the DROP DATABASE statement, consider the following:

- In general, we recommend that you don't drop a database that contains an AWS Data Exchange datashare using the DROP DATABASE statement. If you do, the AWS accounts that have access to the datashare lose access. Performing this type of alteration can breach data product terms in AWS Data Exchange.

The following example shows an error when a database that contains an AWS Data Exchange datashare is dropped.

```
DROP DATABASE test_db;
ERROR: Drop of database test_db that contains ADX-managed datashare(s) requires session variable datashare_break_glass_session_var to be set to value 'ce8d280c10ad41'
```

To allow dropping the database, set the following variable and run the DROP DATABASE statement again.

```
SET datashare_break_glass_session_var to 'ce8d280c10ad41';
```

```
DROP DATABASE test_db;
```

In this case, Amazon Redshift generates a random one-time value to set the session variable to allow DROP DATABASE for a database that contains an AWS Data Exchange datashare.

## Examples

The following example drops a database named TICKIT\_TEST:

```
drop database tickit_test;
```

## DROP DATASHARE

Drops a datashare. This command isn't reversible.

Only a superuser or the datashare owner can drop a datashare.

### Required privileges

Following are required privileges for DROP DATASHARE:

- Superuser
- Users with the DROP DATASHARE privilege
- Datashare owner

## Syntax

```
DROP DATASHARE datashare_name;
```

## Parameters

*datashare\_name*

The name of the datashare to be dropped.

## DROP DATASHARE usage notes

When using the DROP DATASHARE statement, consider the following:

- In general, we recommend that you don't drop an AWS Data Exchange datashare using the DROP DATASHARE statement. If you do, the AWS accounts that have access to the datashare lose access. Performing this type of alteration can breach data product terms in AWS Data Exchange.

The following example shows an error when an AWS Data Exchange datashare is dropped.

```
DROP DATASHARE salesshare;
```

```
ERROR: Drop of ADX-managed datashare salesshare requires session variable
datashare_break_glass_session_var to be set to value '620c871f890c49'
```

To allow dropping an AWS Data Exchange datashare, set the following variable and run the DROP DATASHARE statement again.

```
SET datashare_break_glass_session_var to '620c871f890c49';
```

```
DROP DATASHARE salesshare;
```

In this case, Amazon Redshift generates a random one-time value to set the session variable to allow DROP DATASHARE for an AWS Data Exchange datashare.

## Examples

The following example drops a datashare named salesshare.

```
DROP DATASHARE salesshare;
```

## DROP FUNCTION

Removes a user-defined function (UDF) from the database. The function's signature, or list of argument data types, must be specified because multiple functions can exist with the same name but different signatures. You can't drop an Amazon Redshift built-in function.

This command isn't reversible.

## Required privileges

Following are required privileges for DROP FUNCTION:

- Superuser
- Users with the DROP FUNCTION privilege
- Function owner

## Syntax

```
DROP FUNCTION name
([arg_name] arg_type [, ...])
[CASCADE | RESTRICT]
```

## Parameters

*name*

The name of the function to be removed.

*arg\_name*

The name of an input argument. DROP FUNCTION ignores argument names, because only the argument data types are needed to determine the function's identity.

*arg\_type*

The data type of the input argument. You can supply a comma-separated list with a maximum of 32 data types.

CASCADE

Keyword specifying to automatically drop objects that depend on the function, such as views.

To create a view that isn't dependent on a function, include the WITH NO SCHEMA BINDING clause in the view definition. For more information, see [CREATE VIEW \(p. 851\)](#).

RESTRICT

Keyword specifying that if any objects depend on the function, do not drop the function and return a message. This action is the default.

## Examples

The following example drops the function named f\_sqrt:

```
drop function f_sqrt(int);
```

To remove a function that has dependencies, use the CASCADE option, as shown in the following example:

```
drop function f_sqrt(int)cascade;
```

## DROP GROUP

Deletes a user group. This command isn't reversible. This command doesn't delete the individual users in a group.

See [DROP USER](#) to delete an individual user.

## Syntax

```
DROP GROUP name
```

## Parameter

*name*

Name of the user group to delete.

## Example

The following example deletes the GUEST user group:

```
drop group guests;
```

You can't drop a group if the group has any privileges on an object. If you attempt to drop such a group, you will receive the following error.

```
ERROR: group "guest" can't be dropped because the group has a privilege on some object
```

If the group has privileges for an object, first revoke the privileges before dropping the group. The following example revokes all privileges on all tables in the public schema from the GUEST user group, and then drops the group.

```
revoke all on all tables in schema public from group guest;
drop group guests;
```

## DROP IDENTITY PROVIDER

Deletes an identity provider. This command isn't reversible. Only a superuser can drop an identity provider.

### Syntax

```
DROP IDENTITY PROVIDER identity_provider_name [CASCADE]
```

### Parameters

*identity\_provider\_name*

Name of the identity provider to delete.

CASCADE

Deletes users and roles attached to the identity provider, when it is deleted.

### Example

The following example deletes the *oauth\_provider* identity provider.

```
DROP IDENTITY PROVIDER oauth_provider;
```

If you drop the identity provider, some users may not be able to log in or use client tools configured to use the identity provider.

## DROP LIBRARY

Removes a custom Python library from the database. Only the library owner or a superuser can drop a library.

DROP LIBRARY can't be run inside a transaction block (BEGIN ... END). For more information about transactions, see [Serializable isolation \(p. 119\)](#).

This command isn't reversible. The DROP LIBRARY command commits immediately. If a UDF that depends on the library is running concurrently, the UDF might fail, even if the UDF is running within a transaction.

For more information, see [CREATE LIBRARY \(p. 793\)](#).

### Required privileges

Following are required privileges for DROPLIBRARY:

- Superuser
- Users with the DROP LIBRARY privilege
- Library owner

## Syntax

```
DROP LIBRARY library_name
```

## Parameters

*library\_name*

The name of the library.

## DROP MASKING POLICY

Drops a dynamic data masking policy from all databases. You can't drop a masking policy that's still attached to one or more tables. For more information on dynamic data masking, see [Dynamic data masking \(p. 555\)](#).

Superusers and users or roles that have the sys:secadmin role can drop a masking policy.

## Syntax

```
DROP MASKING POLICY policy_name;
```

## Parameters

*policy\_name*

The name of the masking policy to drop.

## DROP MODEL

Removes a model from the database. Only the model owner or a superuser can drop a model.

DROP MODEL also deletes all the associated prediction function that is derived from this model, all Amazon Redshift artifacts related to the model, and all Amazon S3 data related to the model. While the model is still being trained in Amazon SageMaker, DROP MODEL will cancel those operations.

This command isn't reversible. The DROP MODEL command commits immediately.

## Required permissions

Following are required permissions for DROP MODEL:

- Superuser
- Users with the DROP MODEL permission
- Model owner
- Schema owner

## Syntax

```
DROP MODEL [IF EXISTS] model_name
```

## Parameters

### IF EXISTS

A clause that indicates that if the specified schema already exists, the command should make no changes and return a message that the schema exists.

*model\_name*

The name of the model. The model name in a schema must be unique.

## Examples

The following example drops the model demo\_ml.customer\_churn.

```
DROP MODEL demo_ml.customer_churn
```

## DROP MATERIALIZED VIEW

Removes a materialized view.

For more information about materialized views, see [Creating materialized views in Amazon Redshift \(p. 198\)](#).

## Syntax

```
DROP MATERIALIZED VIEW [IF EXISTS] mv_name
```

## Parameters

### IF EXISTS

A clause that specifies to check if the named materialized view exists. If the materialized view doesn't exist, then the DROP MATERIALIZED VIEW command returns an error message. This clause is useful when scripting, to keep the script from failing if you drop a nonexistent materialized view.

*mv\_name*

The name of the materialized view to be dropped.

## Usage Notes

Only the owner of a materialized view can use DROP MATERIALIZED VIEW on that view. A superuser or a user who has specifically been granted DROP privileges can be exceptions to this.

When you write a drop statement for a materialized view and a view with a matching name exists, it results in an error that instructs you to use DROP VIEW. An error occurs even in a case where you use DROP MATERIALIZED VIEW IF EXISTS.

## Example

The following example drops the `tickets_mv` materialized view.

```
DROP MATERIALIZED VIEW tickets_mv;
```

## DROP PROCEDURE

Drops a procedure. To drop a procedure, both the procedure name and input argument data types (signature), are required. Optionally, you can include the full argument data types, including OUT arguments.

### Required privileges

Following are required privileges for DROP PROCEDURE:

- Superuser
- Users with the DROP PROCEDURE privilege
- Procedure owner

### Syntax

```
DROP PROCEDURE sp_name ([[argname] [argmode] argtype [, ...]])
```

### Parameters

*sp\_name*

The name of the procedure to be removed.

*argname*

The name of an input argument. DROP PROCEDURE ignores argument names, because only the argument data types are needed to determine the procedure's identity.

*argmode*

The mode of an argument, which can be IN, OUT, or INOUT. OUT arguments are optional because they aren't used to identify a stored procedure.

*argtype*

The data type of the input argument. For a list of the supported data types, see [Data types \(p. 574\)](#).

### Examples

The following example drops a stored procedure named `quarterly_revenue`.

```
DROP PROCEDURE quarterly_revenue(volume INOUT bigint, at_price IN numeric,result OUT int);
```

## DROP RLS POLICY

Drops a row-level security policy for all tables in all databases.

Superusers and users or roles that have the sys:secadmin role can drop a policy.

## Syntax

```
DROP RLS POLICY [IF EXISTS] policy_name [CASCADE | RESTRICT]
```

## Parameters

### *IF EXISTS*

A clause that indicates if the specified policy already exists.

### *policy\_name*

The name of the policy.

### *CASCADE*

A clause that indicates to automatically detach the policy from all attached tables before dropping the policy.

### *RESTRICT*

A clause that indicates not to drop the policy when it is attached to some tables. This is the default.

## Examples

The following example drops the row-level security policy.

```
DROP RLS POLICY policy_concerts;
```

## DROP ROLE

Removes a role from a database. Only the role owner who either created the role, a user with the WITH ADMIN option, or a superuser can drop a role.

You can't drop a role that is granted to a user or another role that is dependent on this role.

## Required privileges

Following are the required privileges for DROP ROLE:

- Superuser
- Role owner who is either the user that created the role or a user that has been granted the role with the WITH ADMIN OPTION privilege.

## Syntax

```
DROP ROLE role_name [FORCE | RESTRICT]
```

## Parameters

### *role\_name*

The name of the role.

[ FORCE | RESTRICT ]

The default setting is RESTRICT. Amazon Redshift throws an error when you try to drop a role that has inherited another role. Use FORCE to remove all role assignments, if any exists.

## Examples

The following example drops the role sample\_role.

```
DROP ROLE sample_role FORCE;
```

The following example attempts to drop the role sample\_role1 that has been granted to a user with the default RESTRICT option.

```
CREATE ROLE sample_role1;
GRANT sample_role1 TO user1;
DROP ROLE sample_role1;
ERROR: cannot drop this role since it has been granted on a user
```

To successfully drop the sample\_role1 that has been granted to a user, use the FORCE option.

```
DROP ROLE sample_role1 FORCE;
```

The following example attempts to drop the role sample\_role2 that has another role dependent on it with the default RESTRICT option.

```
CREATE ROLE sample_role1;
CREATE ROLE sample_role2;
GRANT sample_role1 TO sample_role2;
DROP ROLE sample_role2;
ERROR: cannot drop this role since it depends on another role
```

To successfully drop the sample\_role2 that has another role dependent on it, use the FORCE option.

```
DROP ROLE sample_role2 FORCE;
```

## DROP SCHEMA

Deletes a schema. For an external schema, you can also drop the external database associated with the schema. This command isn't reversible.

### Required privileges

Following are required privileges for DROP SCHEMA:

- Superuser
- Schema owner
- Users with the DROP SCHEMA privilege

## Syntax

```
DROP SCHEMA [IF EXISTS] name [, ...]
```

```
[DROP EXTERNAL DATABASE]
[CASCADE | RESTRICT]
```

## Parameters

### IF EXISTS

Clause that indicates that if the specified schema doesn't exist, the command should make no changes and return a message that the schema doesn't exist, rather than terminating with an error.

This clause is useful when scripting, so the script doesn't fail if DROP SCHEMA runs against a nonexistent schema.

### *name*

Names of the schemas to drop. You can specify multiple schema names separated by commas.

### DROP EXTERNAL DATABASE

Clause that indicates that if an external schema is dropped, drop the external database associated with the external schema, if one exists. If no external database exists, the command returns a message stating that no external database exists. If multiple external schemas are dropped, all databases associated with the specified schemas are dropped.

If an external database contains dependent objects such as tables, include the CASCADE option to drop the dependent objects as well.

When you drop an external database, the database is also dropped for any other external schemas associated with the database. Tables defined in other external schemas using the database are also dropped.

DROP EXTERNAL DATABASE doesn't support external databases stored in a HIVE metastore.

### CASCADE

Keyword that indicates to automatically drop all objects in the schema. If DROP EXTERNAL DATABASE is specified, all objects in the external database are also dropped.

### RESTRICT

Keyword that indicates not to drop a schema or external database if it contains any objects. This action is the default.

## Example

The following example deletes a schema named S\_SALES. This example uses RESTRICT as a safety mechanism so that the schema isn't deleted if it contains any objects. In this case, you need to delete the schema objects before deleting the schema.

```
drop schema s_sales restrict;
```

The following example deletes a schema named S\_SALES and all objects that depend on that schema.

```
drop schema s_sales cascade;
```

The following example either drops the S\_SALES schema if it exists, or does nothing and returns a message if it doesn't.

```
drop schema if exists s_sales;
```

The following example deletes an external schema named S\_SPECTRUM and the external database associated with it. This example uses RESTRICT so that the schema and database aren't deleted if they contain any objects. In this case, you need to delete the dependent objects before deleting the schema and the database.

```
drop schema s_spectrum drop external database restrict;
```

The following example deletes multiple schemas and the external databases associated with them, along with any dependent objects.

```
drop schema s_sales, s_profit, s_revenue drop external database cascade;
```

## DROP TABLE

Removes a table from a database.

If you are trying to empty a table of rows, without removing the table, use the DELETE or TRUNCATE command.

DROP TABLE removes constraints that exist on the target table. Multiple tables can be removed with a single DROP TABLE command.

DROP TABLE with an external table can't be run inside a transaction (BEGIN ... END). For more information about transactions, see [Serializable isolation \(p. 119\)](#).

To find an example where the DROP privilege is granted to a group, see GRANT [Examples \(p. 898\)](#).

### Required privileges

Following are required privileges for DROP TABLE:

- Superuser
- Users with the DROP TABLE privilege
- Table owner with the USAGE privilege on the schema

## Syntax

```
DROP TABLE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

### Parameters

#### IF EXISTS

Clause that indicates that if the specified table doesn't exist, the command should make no changes and return a message that the table doesn't exist, rather than terminating with an error.

This clause is useful when scripting, so the script doesn't fail if DROP TABLE runs against a nonexistent table.

*name*

Name of the table to drop.

## CASCADE

Clause that indicates to automatically drop objects that depend on the view, such as other views.

To create a view that isn't dependent on other database objects, such as views and tables, include the WITH NO SCHEMA BINDING clause in the view definition. For more information, see [CREATE VIEW \(p. 851\)](#).

## RESTRICT

Clause that indicates not to drop the table if any objects depend on it. This action is the default.

## Examples

### Dropping a table with no dependencies

The following example creates and drops a table called FEEDBACK that has no dependencies:

```
create table feedback(a int);
drop table feedback;
```

If a table contains columns that are referenced by views or other tables, Amazon Redshift displays a message such as the following.

```
Invalid operation: cannot drop table feedback because other objects depend on it
```

### Dropping two tables simultaneously

The following command set creates a FEEDBACK table and a BUYERS table and then drops both tables with a single command:

```
create table feedback(a int);
create table buyers(a int);
drop table feedback, buyers;
```

### Dropping a table with a dependency

The following steps show how to drop a table called FEEDBACK using the CASCADE switch.

First, create a simple table called FEEDBACK using the CREATE TABLE command:

```
create table feedback(a int);
```

Next, use the CREATE VIEW command to create a view called FEEDBACK\_VIEW that relies on the table FEEDBACK:

```
create view feedback_view as select * from feedback;
```

The following example drops the table FEEDBACK and also drops the view FEEDBACK\_VIEW, because FEEDBACK\_VIEW is dependent on the table FEEDBACK:

```
drop table feedback cascade;
```

### Viewing the dependencies for a table

You can create a view that holds the dependency information for all of the tables in a database. Before dropping a given table, query this view to determine if the table has dependencies.

Type the following command to create a FIND\_DEPEND view, which joins dependencies with object references:

```
create view find_depend as
select distinct c_p.oid as tbloid,
n_p.nspname as schemaname, c_p.relname as name,
n_c.nspname as refbyschemaname, c_c.relname as refbyname,
c_c.oid as viewoid
from pg_catalog.pg_class c_p
join pg_catalog.pg_depend d_p
on c_p.relfilenode = d_p.refobjid
join pg_catalog.pg_depend d_c
on d_p.objid = d_c.objid
join pg_catalog.pg_class c_c
on d_c.refobjid = c_c.relfilenode
left outer join pg_namespace n_p
on c_p.relnamespace = n_p.oid
left outer join pg_namespace n_c
on c_c.relnamespace = n_c.oid
where d_c.deptype = 'i'::"char"
and c_c.relkind = 'v'::"char";
```

Now create a SALES\_VIEW from the SALES table:

```
create view sales_view as select * from sales;
```

Now query the FIND\_DEPEND view to view dependencies in the database. Limit the scope of the query to the PUBLIC schema, as shown in the following code:

```
select * from find_depend
where refbyschemaname='public'
order by name;
```

This query returns the following dependencies, showing that the SALES\_VIEW view is also dropped by using the CASCADE option when dropping the SALES table:

| tbloid | schemaname | name        | viewoid | refbyschemaname | refbyname   |
|--------|------------|-------------|---------|-----------------|-------------|
| 100241 | public     | find_depend | 100241  | public          | find_depend |
| 100203 | public     | sales       | 100245  | public          | sales_view  |
| 100245 | public     | sales_view  | 100245  | public          | sales_view  |

(3 rows)

### Dropping a table Using IF EXISTS

The following example either drops the FEEDBACK table if it exists, or does nothing and returns a message if it doesn't:

```
drop table if exists feedback;
```

## DROP USER

Drops a user from a database. Multiple users can be dropped with a single DROP USER command. You must be a database superuser to run this command.

## Syntax

```
DROP USER [IF EXISTS] name [, ...]
```

## Parameters

### IF EXISTS

Clause that indicates that if the specified user doesn't exist, the command should make no changes and return a message that the user doesn't exist, rather than terminating with an error.

This clause is useful when scripting, so the script doesn't fail if DROP USER runs against a nonexistent user.

### *name*

Name of the user to remove. You can specify multiple users, with a comma separating each user name from the next.

## Usage notes

You can't drop a user if the user owns any database object, such as a schema, database, table, or view, or if the user has any privileges on a database, table, column, or group. If you attempt to drop such a user, you receive one of the following errors.

```
ERROR: user "username" can't be dropped because the user owns some object [SQL State=55006]
```

```
ERROR: user "username" can't be dropped because the user has a privilege on some object [SQL State=55006]
```

### Note

Amazon Redshift checks only the current database before dropping a user. DROP USER doesn't return an error if the user owns database objects or has any privileges on objects in another database. If you drop a user that owns objects in another database, the owner for those objects is changed to 'unknown'.

If a user owns an object, first drop the object or change its ownership to another user before dropping the original user. If the user has privileges for an object, first revoke the privileges before dropping the user. The following example shows dropping an object, changing ownership, and revoking privileges before dropping the user.

```
drop database dwdatabase;
alter schema dw owner to dwadmin;
revoke all on table dwtable from dwuser;
drop user dwuser;
```

## Examples

The following example drops a user called paulo:

```
drop user paulo;
```

The following example drops two users, paulo and martha:

```
drop user paulo, martha;
```

The following example drops the user paulo if it exists, or does nothing and returns a message if it doesn't:

```
drop user if exists paulo;
```

## DROP VIEW

Removes a view from the database. Multiple views can be dropped with a single DROP VIEW command. This command isn't reversible.

### Required privileges

Following are required privileges for DROP VIEW:

- Superuser
- Users with the DROP VIEW privilege
- View owner

### Syntax

```
DROP VIEW [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

### Parameters

#### IF EXISTS

Clause that indicates that if the specified view doesn't exist, the command should make no changes and return a message that the view doesn't exist, rather than terminating with an error.

This clause is useful when scripting, so the script doesn't fail if DROP VIEW runs against a nonexistent view.

#### *name*

Name of the view to be removed.

#### CASCADE

Clause that indicates to automatically drop objects that depend on the view, such as other views.

To create a view that isn't dependent on other database objects, such as views and tables, include the WITH NO SCHEMA BINDING clause in the view definition. For more information, see [CREATE VIEW \(p. 851\)](#).

#### RESTRICT

Clause that indicates not to drop the view if any objects depend on it. This action is the default.

## Examples

The following example drops the view called *event*:

```
drop view event;
```

To remove a view that has dependencies, use the CASCADE option. For example, say we start with a table called EVENT. We then create the eventview view of the EVENT table, using the CREATE VIEW command, as shown in the following example:

```
create view eventview as
select dateid, eventname, catid
from event where catid = 1;
```

Now, we create a second view called *myeventview*, that is based on the first view *eventview*:

```
create view myeventview as
select eventname, catid
from eventview where eventname <> ' ';
```

At this point, two views have been created: *eventview* and *myeventview*.

The *myeventview* view is a child view with *eventview* as its parent.

To delete the *eventview* view, the obvious command to use is the following:

```
drop view eventview;
```

Notice that if you run this command in this case, you get the following error:

```
drop view eventview;
ERROR: can't drop view eventview because other objects depend on it
HINT: Use DROP ... CASCADE to drop the dependent objects too.
```

To remedy this, run the following command (as suggested in the error message):

```
drop view eventview cascade;
```

Both *eventview* and *myeventview* have now been dropped successfully.

The following example either drops the *eventview* view if it exists, or does nothing and returns a message if it doesn't:

```
drop view if exists eventview;
```

## END

Commits the current transaction. Performs exactly the same function as the COMMIT command.

See [COMMIT \(p. 676\)](#) for more detailed documentation.

## Syntax

```
END [WORK | TRANSACTION]
```

## Parameters

WORK

Optional keyword.

## TRANSACTION

Optional keyword; WORK and TRANSACTION are synonyms.

## Examples

The following examples all end the transaction block and commit the transaction:

```
end;
```

```
end work;
```

```
end transaction;
```

After any of these commands, Amazon Redshift ends the transaction block and commits the changes.

## EXECUTE

Runs a previously prepared statement.

## Syntax

```
EXECUTE plan_name [(parameter [, ...])]
```

## Parameters

*plan\_name*

Name of the prepared statement to be run.

*parameter*

The actual value of a parameter to the prepared statement. This must be an expression yielding a value of a type compatible with the data type specified for this parameter position in the PREPARE command that created the prepared statement.

## Usage notes

EXECUTE is used to run a previously prepared statement. Because prepared statements only exist for the duration of a session, the prepared statement must have been created by a PREPARE statement run earlier in the current session.

If the previous PREPARE statement specified some parameters, a compatible set of parameters must be passed to the EXECUTE statement, or else Amazon Redshift returns an error. Unlike functions, prepared statements aren't overloaded based on the type or number of specified parameters; the name of a prepared statement must be unique within a database session.

When an EXECUTE command is issued for the prepared statement, Amazon Redshift may optionally revise the query execution plan (to improve performance based on the specified parameter values) before running the prepared statement. Also, for each new execution of a prepared statement, Amazon Redshift may revise the query execution plan again based on the different parameter values specified with the EXECUTE statement. To examine the query execution plan that Amazon Redshift has chosen for any given EXECUTE statements, use the [EXPLAIN \(p. 881\)](#) command.

For examples and more information on the creation and usage of prepared statements, see [PREPARE \(p. 912\)](#).

## See also

[DEALLOCATE \(p. 854\)](#), [PREPARE \(p. 912\)](#)

## EXPLAIN

Displays the execution plan for a query statement without running the query.

### Syntax

```
EXPLAIN [VERBOSE] query
```

### Parameters

#### VERBOSE

Displays the full query plan instead of just a summary.

#### *query*

Query statement to explain. The query can be a SELECT, INSERT, CREATE TABLE AS, UPDATE, or DELETE statement.

## Usage notes

EXPLAIN performance is sometimes influenced by the time it takes to create temporary tables. For example, a query that uses the common subexpression optimization requires temporary tables to be created and analyzed in order to return the EXPLAIN output. The query plan depends on the schema and statistics of the temporary tables. Therefore, the EXPLAIN command for this type of query might take longer to run than expected.

You can use EXPLAIN only for the following commands:

- SELECT
- SELECT INTO
- CREATE TABLE AS
- INSERT
- UPDATE
- DELETE

The EXPLAIN command will fail if you use it for other SQL commands, such as data definition language (DDL) or database operations.

## Query planning and execution steps

The execution plan for a specific Amazon Redshift query statement breaks down execution and calculation of a query into a discrete sequence of steps and table operations that eventually produce a final result set for the query. For information about query planning, see [Query processing \(p. 463\)](#).

The following table provides a summary of steps that Amazon Redshift can use in developing an execution plan for any query a user submits for execution.

| <b>EXPLAIN operators</b>                                                                                                                                                                                                                                                                   | <b>Query execution steps</b> | <b>Description</b>                                                                                                                                                                                                                                                                                 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SCAN:</b>                                                                                                                                                                                                                                                                               |                              |                                                                                                                                                                                                                                                                                                    |
| Sequential Scan                                                                                                                                                                                                                                                                            | scan                         | Amazon Redshift relation scan or table scan operator or step. Scans whole table sequentially from beginning to end; also evaluates query constraints for every row (Filter) if specified with WHERE clause. Also used to run INSERT, UPDATE, and DELETE statements.                                |
| <b>JOINS:</b> Amazon Redshift uses different join operators based on the physical design of the tables being joined, the location of the data required for the join, and specific attributes of the query itself. Subquery Scan -- Subquery scan and append are used to run UNION queries. |                              |                                                                                                                                                                                                                                                                                                    |
| Nested Loop                                                                                                                                                                                                                                                                                | nloop                        | Least optimal join; mainly used for cross-joins (Cartesian products; without a join condition) and some inequality joins.                                                                                                                                                                          |
| Hash Join                                                                                                                                                                                                                                                                                  | hjoin                        | Also used for inner joins and left and right outer joins and typically faster than a nested loop join. Hash Join reads the outer table, hashes the joining column, and finds matches in the inner hash table. Step can spill to disk. (Inner input of hjoin is hash step which can be disk-based.) |
| Merge Join                                                                                                                                                                                                                                                                                 | mjoin                        | Also used for inner joins and outer joins (for join tables that are both distributed and sorted on the joining columns). Typically the fastest Amazon Redshift join algorithm, not including other cost considerations.                                                                            |
| <b>AGGREGATION:</b> Operators and steps used for queries that involve aggregate functions and GROUP BY operations.                                                                                                                                                                         |                              |                                                                                                                                                                                                                                                                                                    |
| Aggregate                                                                                                                                                                                                                                                                                  | aggr                         | Operator/step for scalar aggregate functions.                                                                                                                                                                                                                                                      |
| HashAggregate                                                                                                                                                                                                                                                                              | aggr                         | Operator/step for grouped aggregate functions. Can operate from disk by virtue of hash table spilling to disk.                                                                                                                                                                                     |
| GroupAggregate                                                                                                                                                                                                                                                                             | aggr                         | Operator sometimes chosen for grouped aggregate queries if the Amazon Redshift configuration setting for force_hash_grouping setting is off.                                                                                                                                                       |
| <b>SORT:</b> Operators and steps used when queries have to sort or merge result sets.                                                                                                                                                                                                      |                              |                                                                                                                                                                                                                                                                                                    |
| Sort                                                                                                                                                                                                                                                                                       | sort                         | Sort performs the sorting specified by the ORDER BY clause as well as other operations such as UNIONs and joins. Can operate from disk.                                                                                                                                                            |
| Merge                                                                                                                                                                                                                                                                                      | merge                        | Produces final sorted results of a query based on intermediate sorted results derived from operations performed in parallel.                                                                                                                                                                       |

| <b>EXPLAIN operators</b>                            | <b>Query execution steps</b> | <b>Description</b>                                                                                                                                                                                                                                                                       |
|-----------------------------------------------------|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>EXCEPT, INTERSECT, and UNION operations:</b>     |                              |                                                                                                                                                                                                                                                                                          |
| SetOp Except [Distinct]                             | hjoin                        | Used for EXCEPT queries. Can operate from disk based on virtue of fact that input hash can be disk-based.                                                                                                                                                                                |
| Hash Intersect [Distinct]                           | hjoin                        | Used for INTERSECT queries. Can operate from disk based on virtue of fact that input hash can be disk-based.                                                                                                                                                                             |
| Append [All  Distinct]                              | save                         | Append used with Subquery Scan to implement UNION and UNION ALL queries. Can operate from disk based on virtue of "save".                                                                                                                                                                |
| <b>Miscellaneous/Other:</b>                         |                              |                                                                                                                                                                                                                                                                                          |
| Hash                                                | hash                         | Used for inner joins and left and right outer joins (provides input to a hash join). The Hash operator creates the hash table for the inner table of a join. (The inner table is the table that is checked for matches and, in a join of two tables, is usually the smaller of the two.) |
| Limit                                               | limit                        | Evaluates the LIMIT clause.                                                                                                                                                                                                                                                              |
| Materialize                                         | save                         | Materialize rows for input to nested loop joins and some merge joins. Can operate from disk.                                                                                                                                                                                             |
| --                                                  | parse                        | Used to parse textual input data during a load.                                                                                                                                                                                                                                          |
| --                                                  | project                      | Used to rearrange columns and compute expressions, that is, project data.                                                                                                                                                                                                                |
| Result                                              | --                           | Run scalar functions that don't involve any table access.                                                                                                                                                                                                                                |
| --                                                  | return                       | Return rows to the leader or client.                                                                                                                                                                                                                                                     |
| Subplan                                             | --                           | Used for certain subqueries.                                                                                                                                                                                                                                                             |
| Unique                                              | unique                       | Eliminates duplicates from SELECT DISTINCT and UNION queries.                                                                                                                                                                                                                            |
| Window                                              | window                       | Compute aggregate and ranking window functions. Can operate from disk.                                                                                                                                                                                                                   |
| <b>Network Operations:</b>                          |                              |                                                                                                                                                                                                                                                                                          |
| Network (Broadcast)                                 | bcast                        | Broadcast is also an attribute of Join Explain operators and steps.                                                                                                                                                                                                                      |
| Network (Distribute)                                | dist                         | Distribute rows to compute nodes for parallel processing by data warehouse cluster.                                                                                                                                                                                                      |
| Network (Send to Leader)                            | return                       | Sends results back to the leader for further processing.                                                                                                                                                                                                                                 |
| <b>DML Operations (operators that modify data):</b> |                              |                                                                                                                                                                                                                                                                                          |

| EXPLAIN operators      | Query execution steps | Description                          |
|------------------------|-----------------------|--------------------------------------|
| Insert (using Result)  | insert                | Inserts data.                        |
| Delete (Scan + Filter) | delete                | Deletes data. Can operate from disk. |
| Update (Scan + Filter) | delete, insert        | Implemented as delete and Insert.    |

## Using EXPLAIN for RLS

If a query contains a table that is subject to row-level security (RLS) policies, EXPLAIN displays a special RLS SecureScan node. Amazon Redshift also logs the same node type to the STL\_EXPLAIN system table. EXPLAIN doesn't reveal the RLS predicate that applies to dim\_tbl. The RLS SecureScan node type serves as an indicator that the execution plan contains additional operations that are invisible to the current user.

The following example illustrates an RLS SecureScan node.

```
EXPLAIN
SELECT D.cint
FROM fact_tbl F INNER JOIN dim_tbl D ON F.k_dim = D.k
WHERE F.k_dim / 10 > 0;
 QUERY PLAN

XN Hash Join DS_DIST_ALL_NONE (cost=0.08..0.25 rows=1 width=4)
 Hash Cond: ("outer".k_dim = "inner"."k")
 -> *XN* *RLS SecureScan f (cost=0.00..0.14 rows=2 width=4)*
 Filter: ((k_dim / 10) > 0)
 -> XN Hash (cost=0.07..0.07 rows=2 width=8)
 -> XN Seq Scan on dim_tbl d (cost=0.00..0.07 rows=2 width=8)
 Filter: ((k / 10) > 0)
```

To enable full investigation of query plans that are subject to RLS, Amazon Redshift offers the EXPLAIN RLS system permissions. Users that have been granted this permission can inspect complete query plans that also include RLS predicates.

The following example illustrates an additional Seq Scan below the RLS SecureScan node also includes the RLS policy predicate ( $k\_dim > 1$ ).

```
EXPLAIN SELECT D.cint
FROM fact_tbl F INNER JOIN dim_tbl D ON F.k_dim = D.k
WHERE F.k_dim / 10 > 0;
 QUERY PLAN

XN Hash Join DS_DIST_ALL_NONE (cost=0.08..0.25 rows=1 width=4)
 Hash Cond: ("outer".k_dim = "inner"."k")
 *-> XN RLS SecureScan f (cost=0.00..0.14 rows=2 width=4)
 Filter: ((k_dim / 10) > 0)*
 -> *XN* *Seq Scan on fact_tbl rls_table (cost=0.00..0.06 rows=5 width=8)
 Filter: (k_dim > 1)*
 -> XN Hash (cost=0.07..0.07 rows=2 width=8)
 -> XN Seq Scan on dim_tbl d (cost=0.00..0.07 rows=2 width=8)
 Filter: ((k / 10) > 0)
```

While the EXPLAIN RLS permission is granted to a user, Amazon Redshift logs the full query plan including RLS predicates in the STL\_EXPLAIN system table. Queries that are run while this permission is not granted will be logged without RLS internals. Granting or removing the EXPLAIN RLS permission won't change what Amazon Redshift has logged to STL\_EXPLAIN for previous queries.

## AWS Lake Formation-RLS protected Redshift relations

The following example illustrates an LF SecureScan node, which you can use to view Lake Formation-RLS relations.

```
EXPLAIN
SELECT *
FROM lf_db.public.t_share
WHERE a > 1;
QUERY PLAN

XN LF SecureScan t_share (cost=0.00..0.02 rows=2 width=11)
(2 rows)
```

## Examples

### Note

For these examples, the sample output might vary depending on Amazon Redshift configuration.

The following example returns the query plan for a query that selects the EVENTID, EVENTNAME, VENUEID, and VENUENAME from the EVENT and VENUE tables:

```
explain
select eventid, eventname, event.venueid, venuename
from event, venue
where event.venueid = venue.venueid;
```

```

QUERY PLAN

XN Hash Join DS_DIST_OUTER (cost=2.52..58653620.93 rows=8712 width=43)
Hash Cond: ("outer".venueid = "inner".venueid)
-> XN Seq Scan on event (cost=0.00..87.98 rows=8798 width=23)
-> XN Hash (cost=2.02..2.02 rows=202 width=22)
-> XN Seq Scan on venue (cost=0.00..2.02 rows=202 width=22)
(5 rows)
```

The following example returns the query plan for the same query with verbose output:

```
explain verbose
select eventid, eventname, event.venueid, venuename
from event, venue
where event.venueid = venue.venueid;
```

```

QUERY PLAN

{HASHJOIN
:startup_cost 2.52
:total_cost 58653620.93
:plan_rows 8712
:plan_width 43
:best_pathkeys <>
:dist_info DS_DIST_OUTER
:dist_info.dist_keys (
TARGETENTRY
{
VAR
:varno 2
:varattno 1
```

```
...
XN Hash Join DS_DIST_OUTER (cost=2.52..58653620.93 rows=8712 width=43)
Hash Cond: ("outer".venueid = "inner".venueid)
-> XN Seq Scan on event (cost=0.00..87.98 rows=8798 width=23)
-> XN Hash (cost=2.02..2.02 rows=202 width=22)
-> XN Seq Scan on venue (cost=0.00..2.02 rows=202 width=22)
(519 rows)
```

The following example returns the query plan for a CREATE TABLE AS (CTAS) statement:

```
explain create table venue_nonulls as
select * from venue
where venueSeats is not null;

QUERY PLAN

XN Seq Scan on venue (cost=0.00..2.02 rows=187 width=45)
Filter: (venueSeats IS NOT NULL)
(2 rows)
```

## FETCH

Retrieves rows using a cursor. For information about declaring a cursor, see [DECLARE \(p. 855\)](#).

FETCH retrieves rows based on the current position within the cursor. When a cursor is created, it is positioned before the first row. After a FETCH, the cursor is positioned on the last row retrieved. If FETCH runs off the end of the available rows, such as following a FETCH ALL, the cursor is left positioned after the last row.

FORWARD 0 fetches the current row without moving the cursor; that is, it fetches the most recently fetched row. If the cursor is positioned before the first row or after the last row, no row is returned.

When the first row of a cursor is fetched, the entire result set is materialized on the leader node, in memory or on disk, if needed. Because of the potential negative performance impact of using cursors with large result sets, we recommend using alternative approaches whenever possible. For more information, see [Performance considerations when using cursors \(p. 856\)](#).

For more information, see [DECLARE \(p. 855\)](#), [CLOSE \(p. 674\)](#).

## Syntax

```
FETCH [NEXT | ALL | {FORWARD [count | ALL] }] FROM cursor
```

## Parameters

**NEXT**

Fetches the next row. This is the default.

**ALL**

Fetches all remaining rows. (Same as FORWARD ALL.) ALL isn't supported for single-node clusters.

**FORWARD [ count | ALL ]**

Fetches the next *count* rows, or all remaining rows. FORWARD 0 fetches the current row. For single-node clusters, the maximum value for count is 1000. FORWARD ALL isn't supported for single-node clusters.

*cursor*

Name of the new cursor.

## FETCH example

The following example declares a cursor named LOLLAPALOOZA to select sales information for the Lollapalooza event, and then fetches rows from the result set using the cursor:

```
-- Begin a transaction
begin;

-- Declare a cursor

declare lollapalooza cursor for
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Lollapalooza';

-- Fetch the first 5 rows in the cursor lollapalooza:

fetch forward 5 from lollapalooza;

eventname | starttime | costperticket | qtysold
-----+-----+-----+-----+
Lollapalooza | 2008-05-01 19:00:00 | 92.00000000 | 3
Lollapalooza | 2008-11-15 15:00:00 | 222.00000000 | 2
Lollapalooza | 2008-04-17 15:00:00 | 239.00000000 | 3
Lollapalooza | 2008-04-17 15:00:00 | 239.00000000 | 4
Lollapalooza | 2008-04-17 15:00:00 | 239.00000000 | 1
(5 rows)

-- Fetch the next row:

fetch next from lollapalooza;

eventname | starttime | costperticket | qtysold
-----+-----+-----+-----+
Lollapalooza | 2008-10-06 14:00:00 | 114.00000000 | 2

-- Close the cursor and end the transaction:

close lollapalooza;
commit;
```

## GRANT

Defines access permissions for a user or user group.

Permissions include access options such as being able to read data in tables and views, write data, create tables, and drop tables. Use this command to give specific permissions for a table, database, schema, function, procedure, language, or column. To revoke permissions from a database object, use the [REVOKE \(p. 916\)](#) command.

Permissions also include access options such as being able to add objects or consumers to or remove objects or consumers from a datashare. To add database objects to or remove database objects from a datashare for a user or user group, use the ALTER permission. Similarly, to add or remove consumers from a datashare, use the SHARE permission. To revoke permissions from a database object, use the

[REVOKE \(p. 916\)](#) command. ALTER and SHARE are the only permissions that you can grant to users and user groups.

You can also grant roles to manage database permissions and control what users can do relative to your data. By defining roles and assigning roles to users, you can limit the actions those users can take, such as limiting users to only the CREATE TABLE and INSERT commands. For more information about the CREATE ROLE command, see [the section called "CREATE ROLE" \(p. 820\)](#). Amazon Redshift has some system-defined roles that you can also use to grant specific permissions to your users. For more information, see [the section called "Amazon Redshift system-defined roles" \(p. 542\)](#).

You can only GRANT or REVOKE USAGE permissions on an external schema to database users and user groups that use the ON SCHEMA syntax. When using ON EXTERNAL SCHEMA with AWS Lake Formation, you can only GRANT and REVOKE permissions to an AWS Identity and Access Management (IAM) role. For the list of permissions, see the syntax.

For stored procedures, the only permission that you can grant is EXECUTE.

You can't run GRANT (on an external resource) within a transaction block (BEGIN ... END). For more information about transactions, see [Serializable isolation \(p. 119\)](#).

## Syntax

```

GRANT { { SELECT | INSERT | UPDATE | DELETE | DROP | REFERENCES } [, ...] | ALL
[PRIVILEGES] }
 ON { [TABLE] table_name [, ...] | ALL TABLES IN SCHEMA schema_name [, ...] }
 TO { username [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC }
[, ...]

GRANT { { CREATE | TEMPORARY | TEMP } [, ...] | ALL [PRIVILEGES] }
 ON DATABASE db_name [, ...]
 TO { username [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC }
[, ...]

GRANT { { CREATE | USAGE } [, ...] | ALL [PRIVILEGES] }
 ON SCHEMA schema_name [, ...]
 TO { username [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC }
[, ...]

GRANT { EXECUTE | ALL [PRIVILEGES] }
 ON { FUNCTION function_name ([[argname] argtype [, ...]]) [, ...] | ALL FUNCTIONS
IN SCHEMA schema_name [, ...] }
 TO { username [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC }
[, ...]

GRANT { EXECUTE | ALL [PRIVILEGES] }
 ON { PROCEDURE procedure_name ([[argname] argtype [, ...]]) [, ...] | ALL
PROCEDURES IN SCHEMA schema_name [, ...] }
 TO { username [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC }
[, ...]

GRANT USAGE
 ON LANGUAGE language_name [, ...]
 TO { username [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC }
[, ...]

```

### Granting column-level permissions for tables

The following is the syntax for column-level permissions on Amazon Redshift tables and views.

```

GRANT { { SELECT | UPDATE } (column_name [, ...]) [, ...] | ALL [PRIVILEGES]
(column_name [, ...]) }

```

```
ON { [TABLE] table_name [, ...] }
TO { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

### Granting ASSUMEROLE permissions

The following is the syntax for the ASSUMEROLE permissions granted to users and groups with a specified role. To begin using the ASSUMEROLE privilege, see [Usage notes for granting the ASSUMEROLE permission \(p. 897\)](#).

```
GRANT ASSUMEROLE
 ON { 'iam_role' [, ...] | default | ALL }
 TO { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
 FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL } [, ...]
```

### Granting permissions for Redshift Spectrum integration with Lake Formation

The following is the syntax for Redshift Spectrum integration with Lake Formation.

```
GRANT { SELECT | ALL [PRIVILEGES] } (column_list)
 ON EXTERNAL TABLE schema_name.table_name
 TO { IAM_ROLE iam_role } [, ...] [WITH GRANT OPTION]

GRANT { { SELECT | ALTER | DROP | DELETE | INSERT } [, ...] | ALL [PRIVILEGES] }
 ON EXTERNAL TABLE schema_name.table_name [, ...]
 TO { { IAM_ROLE iam_role } [, ...] | PUBLIC } [WITH GRANT OPTION]

GRANT { { CREATE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
 ON EXTERNAL SCHEMA schema_name [, ...]
 TO { IAM_ROLE iam_role } [, ...] [WITH GRANT OPTION]
```

### Granting datashare permissions

The following is the syntax for using GRANT for datashare permissions on Amazon Redshift. ALTER and SHARE are the only permissions that you can grant to users and user groups in this case.

```
GRANT { ALTER | SHARE } ON DATASHARE datashare_name
 TO { username [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

The following is the syntax for using GRANT for datashare usage permissions on Amazon Redshift. You grant access to a datashare to a consumer using the USAGE permission. You can't grant this permission to users or user groups. This permission also doesn't support the WITH GRANT OPTION for the GRANT statement. Only users or user groups with the SHARE permission previously granted to them FOR the datashare can run this type of GRANT statement.

```
GRANT USAGE
 ON DATASHARE datashare_name
 TO NAMESPACE 'namespaceGUID' [, ...] | ACCOUNT 'accountnumber' [VIA DATA CATALOG]
 [, ...]
```

The following is an example of how to grant usage of a datashare to a Lake Formation account.

```
GRANT USAGE ON salesshare TO ACCOUNT '123456789012' VIA DATA CATALOG;
```

The following is the syntax for GRANT data-sharing usage permissions on a specific database or schema created from a datashare. This USAGE permission doesn't grant usage permission to databases

that aren't created from the specified datashare. You can only GRANT or REVOKE ALTER or SHARE permissions on a datashare to users and user groups.

```
GRANT USAGE ON { DATABASE shared_database_name [, ...] | SCHEMA shared_schema}
 TO { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

### Granting machine learning permissions

The following is the syntax for machine learning model permissions on Amazon Redshift.

```
GRANT CREATE MODEL
 TO { username [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC } [, ...]

GRANT { EXECUTE | ALL [PRIVILEGES] }
 ON MODEL model_name [, ...]

TO { username [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

### Granting role permissions

The following is the syntax for granting role permissions on Amazon Redshift.

```
GRANT { ROLE role_name } [, ...] TO { { user_name [WITH ADMIN OPTION] } | ROLE role_name } [, ...]
```

The following is the syntax for granting system permissions to roles on Amazon Redshift.

```
GRANT
{
 { CREATE USER | DROP USER | ALTER USER |
 CREATE SCHEMA | DROP SCHEMA |
 ALTER DEFAULT PRIVILEGES |
 CREATE TABLE | DROP TABLE | ALTER TABLE |
 CREATE OR REPLACE FUNCTION | CREATE OR REPLACE EXTERNAL FUNCTION |
 DROP FUNCTION |
 CREATE OR REPLACE PROCEDURE | DROP PROCEDURE |
 CREATE OR REPLACE VIEW | DROP VIEW |
 CREATE MODEL | DROP MODEL |
 CREATE DATASHARE | ALTER DATASHARE | DROP DATASHARE |
 CREATE LIBRARY | DROP LIBRARY |
 CREATE ROLE | DROP ROLE |
 TRUNCATE TABLE |
 VACUUM | ANALYZE | CANCEL } [, ...]
}
| { ALL [PRIVILEGES] }
TO { ROLE role_name } [, ...]
```

### Granting explain permissions for row-level security policy filters

The following is the syntax for granting permissions to explain the row-level security policy filters of a query in the EXPLAIN plan. You can revoke the privilege using the REVOKE statement.

```
GRANT EXPLAIN RLS TO ROLE rolename
```

The following is the syntax for granting permissions to bypass row-level security policies for a query.

```
GRANT IGNORE RLS TO ROLE rolename
```

### Granting permissions for RLS lookup tables to a policy object

The following is the syntax for granting permissions to the specified row-level security policy.

```
GRANT SELECT ON [TABLE] table_name [, ...]
TO RLS POLICY policy_name [, ...]
```

## Parameters

### SELECT

Grants permission to select data from a table or view using a SELECT statement. The SELECT permission is also required to reference existing column values for UPDATE or DELETE operations.

### INSERT

Grants permission to load data into a table using an INSERT statement or a COPY statement.

### UPDATE

Grants permission to update a table column using an UPDATE statement. UPDATE operations also require the SELECT permission, because they must reference table columns to determine which rows to update, or to compute new values for columns.

### DELETE

Grants permission to delete a data row from a table. DELETE operations also require the SELECT permission, because they must reference table columns to determine which rows to delete.

### REFERENCES

Grants permission to create a foreign key constraint. You need to grant this permission on both the referenced table and the referencing table; otherwise, the user can't create the constraint.

### ALL [ PRIVILEGES ]

Grants all available permissions at once to the specified user or user group. The PRIVILEGES keyword is optional.

GRANT ALL ON SCHEMA doesn't grant CREATE permissions for external schemas.

You can grant the ALL permission to a table in an AWS Glue Data Catalog that is enabled for Lake Formation. In this case, individual permissions (such as SELECT, ALTER, and so on) are recorded in the Data Catalog.

### ALTER

Grants permission to alter a table in an AWS Glue Data Catalog that is enabled for Lake Formation. This permission only applies when using Lake Formation.

### DROP

Grants permission to drop a table. This permission applies in Amazon Redshift and in an AWS Glue Data Catalog that is enabled for Lake Formation.

### ASSUMEROLE

Grants permission to run COPY, UNLOAD, EXTERNAL FUNCTION, and CREATE MODEL commands to users, roles, or groups with a specified role. The user, role, or group assumes that role when running the specified command. To begin using the ASSUMEROLE permission, see [Usage notes for granting the ASSUMEROLE permission \(p. 897\)](#).

ON [ TABLE ] *table\_name*

Grants the specified permissions on a table or a view. The TABLE keyword is optional. You can list multiple tables and views in one statement.

ON ALL TABLES IN SCHEMA *schema\_name*

Grants the specified permissions on all tables and views in the referenced schema.

(*column\_name* [...] ) ON TABLE *table\_name*

Grants the specified permissions to users, groups, or PUBLIC on the specified columns of the Amazon Redshift table or view.

(*column\_list* ) ON EXTERNAL TABLE *schema\_name.table\_name*

Grants the specified permissions to an IAM role on the specified columns of the Lake Formation table in the referenced schema.

ON EXTERNAL TABLE *schema\_name.table\_name*

Grants the specified permissions to an IAM role on the specified Lake Formation tables in the referenced schema.

ON EXTERNAL SCHEMA *schema\_name*

Grants the specified permissions to an IAM role on the referenced schema.

ON *iam\_role*

Grants the specified permissions to an IAM role.

TO *username*

Indicates the user receiving the permissions.

TO IAM\_ROLE *iam\_role*

Indicates the IAM role receiving the permissions.

WITH GRANT OPTION

Indicates that the user receiving the permissions can in turn grant the same permissions to others.

WITH GRANT OPTION can't be granted to a group or to PUBLIC.

ROLE *role\_name*

Grants the permissions to a role.

GROUP *group\_name*

Grants the permissions to a user group. Can be a comma-separated list to specify multiple user groups.

PUBLIC

Grants the specified permissions to all users, including users created later. PUBLIC represents a group that always includes all users. An individual user's permissions consist of the sum of permissions granted to PUBLIC, permissions granted to any groups that the user belongs to, and any permissions granted to the user individually.

Granting PUBLIC to a Lake Formation EXTERNAL TABLE results in granting the permission to the Lake Formation *everyone* group.

CREATE

Depending on the database object, grants the following permissions to the user or user group:

- For databases, CREATE allows users to create schemas within the database.
- For schemas, CREATE allows users to create objects within a schema. To rename an object, the user must have the CREATE permission and own the object to be renamed.
- CREATE ON SCHEMA isn't supported for Amazon Redshift Spectrum external schemas. To grant usage of external tables in an external schema, grant USAGE ON SCHEMA to the users that need access. Only the owner of an external schema or a superuser is permitted to create external tables in the external schema. To transfer ownership of an external schema, use [ALTER SCHEMA \(p. 638\)](#) to change the owner.

TEMPORARY | TEMP

Grants the permission to create temporary tables in the specified database. To run Amazon Redshift Spectrum queries, the database user must have permission to create temporary tables in the database.

**Note**

By default, users are granted permission to create temporary tables by their automatic membership in the PUBLIC group. To remove the permission for any users to create temporary tables, revoke the TEMP permission from the PUBLIC group. Then explicitly grant the permission to create temporary tables to specific users or groups of users.

ON DATABASE *db\_name*

Grants the specified permissions on a database.

USAGE

Grants USAGE permission on a specific schema, which makes objects in that schema accessible to users. Specific actions on these objects must be granted separately (for example, SELECT or UPDATE permission on tables) for local Amazon Redshift schemas. By default, all users have CREATE and USAGE permission on the PUBLIC schema.

When you grant USAGE to external schemas using ON SCHEMA syntax, you don't need to grant actions separately on the objects in the external schema. The corresponding catalog permissions control granular permissions on the external schema objects.

ON SCHEMA *schema\_name*

Grants the specified permissions on a schema.

GRANT CREATE ON SCHEMA and the CREATE permission in GRANT ALL ON SCHEMA aren't supported for Amazon Redshift Spectrum external schemas. To grant usage of external tables in an external schema, grant USAGE ON SCHEMA to the users that need access. Only the owner of an external schema or a superuser is permitted to create external tables in the external schema. To transfer ownership of an external schema, use [ALTER SCHEMA \(p. 638\)](#) to change the owner.

EXECUTE ON ALL FUNCTIONS IN SCHEMA *schema\_name*

Grants the specified permissions on all functions in the referenced schema.

Amazon Redshift doesn't support GRANT or REVOKE statements for pg\_proc builtin entries defined in pg\_catalog namespace.

EXECUTE ON PROCEDURE *procedure\_name*

Grants the EXECUTE permission on a specific stored procedure. Because stored procedure names can be overloaded, you must include the argument list for the procedure. For more information, see [Naming stored procedures \(p. 166\)](#).

EXECUTE ON ALL PROCEDURES IN SCHEMA *schema\_name*

Grants the specified permissions on all stored procedures in the referenced schema.

USAGE ON LANGUAGE *language\_name*

Grants the USAGE permission on a language.

The USAGE ON LANGUAGE permission is required to create user-defined functions (UDFs) by running the [CREATE FUNCTION \(p. 788\)](#) command. For more information, see [UDF security and privileges \(p. 150\)](#).

The USAGE ON LANGUAGE permission is required to create stored procedures by running the [CREATE PROCEDURE \(p. 815\)](#) command. For more information, see [Security and privileges for stored procedures \(p. 167\)](#).

For Python UDFs, use plpythonu. For SQL UDFs, use sql. For stored procedures, use plpgsql.

FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL } [, ...]

Specifies the SQL command for which the permission is granted. You can specify ALL to grant the permission on the COPY, UNLOAD, EXTERNAL FUNCTION, and CREATE MODEL statements. This clause applies only to granting the ASSUMEROLE permission.

ALTER

Grants the ALTER permission to users to add or remove objects from a datashare, or to set the property PUBLICACCESSIBLE. For more information, see [ALTER DATASHARE \(p. 629\)](#).

SHARE

Grants permissions to users and user groups to add data consumers to a datashare. This permission is required to enable the particular consumer (account or namespace) to access the datashare from their clusters. The consumer can be the same or a different AWS account, with the same or a different cluster namespace as specified by a globally unique identifier (GUID).

ON DATASHARE *datashare\_name*

Grants the specified permissions on the referenced datashare. For information about consumer access control granularity, see [Sharing data at different levels in Amazon Redshift \(p. 320\)](#).

USAGE

When USAGE is granted to a consumer account or namespace within the same account, the specific consumer account or namespace within the account can access the datashare and the objects of the datashare in read-only fashion.

TO NAMESPACE 'clusternamespace' GUID'

Indicates a namespace in the same account where consumers can receive the specified permissions to the datashare. Namespaces use a 128-bit alphanumeric GUID.

TO ACCOUNT 'accountnumber' [ VIA DATA CATALOG ]

Indicates the number of another account whose consumers can receive the specified permissions to the datashare. Specifying 'VIA DATA CATALOG' indicates that you are granting usage of the datashare to a Lake Formation account. Omitting this parameter means you're granting usage to an account that owns the cluster.

ON DATABASE *shared\_database\_name* [, ...]

Grants the specified usage permissions on the specified database that is created in the specified datashare.

ON SCHEMA *shared\_schema*

Grants the specified permissions on the specified schema that is created in the specified datashare.

CREATE MODEL

Grants the CREATE MODEL permission to specific users or user groups.

ON MODEL *model\_name*

Grants the EXECUTE permission on a specific model.

{ role } [ , ... ]

The role to be granted to another role, a user, or PUBLIC.

PUBLIC represents a group that always includes all users. An individual user's permissions consist of the sum of permissions granted to PUBLIC, permissions granted to any groups that the user belongs to, and any permissions granted to the user individually.

TO { { *user\_name* [ WITH ADMIN OPTION ] } | role }[ , ... ]

Grants the specified role to a specified user with the WITH ADMIN OPTION, another role, or PUBLIC.

The WITH ADMIN OPTION clause provides the administration options for all the granted roles to all the grantees.

EXPLAIN RLS TO ROLE *rolename*

Grants the permission to explain the row-level security policy filters of a query in the EXPLAIN plan to a role.

IGNORE RLS TO ROLE *rolename*

Grants the permission to bypass row-level security policies for a query to a role.

## Usage notes

To learn more about the usage notes for GRANT, see [the section called “Usage notes” \(p. 895\)](#).

## Examples

For examples of how to use GRANT, see [the section called “Examples” \(p. 898\)](#).

## Usage notes

To grant privileges on an object, you must meet one of the following criteria:

- Be the object owner.
- Be a superuser.
- Have a grant privilege for that object and privilege.

For example, the following command enables the user HR both to perform SELECT commands on the employees table and to grant and revoke the same privilege for other users.

```
grant select on table employees to HR with grant option;
```

HR can't grant privileges for any operation other than SELECT, or on any other table than employees.

As another example, the following command enables the user HR both to perform ALTER commands on the employees table and to grant and revoke the same privilege for other users.

```
grant ALTER on table employees to HR with grant option;
```

HR can't grant privileges for any operation other than ALTER, or on any other table than employees.

Having privileges granted on a view doesn't imply having privileges on the underlying tables. Similarly, having privileges granted on a schema doesn't imply having privileges on the tables in the schema. Instead, grant access to the underlying tables explicitly.

To grant privileges to an AWS Lake Formation table, the IAM role associated with the table's external schema must have permission to grant privileges to the external table. The following example creates an external schema with an associated IAM role `myGrantor`. The IAM role `myGrantor` has the permission to grant permissions to others. The GRANT command uses the permission of the IAM role `myGrantor` that is associated with the external schema to grant permission to the IAM role `myGrantee`.

```
create external schema mySchema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myGrantor'
create external database if not exists;
```

```
grant select
on external table mySchema.mytable
to iam_role 'arn:aws:iam::123456789012:role/myGrantee';
```

If you GRANT ALL privileges to an IAM role, individual privileges are granted in the related Lake Formation–enabled Data Catalog. For example, the following GRANT ALL results in the granted individual privileges (SELECT, ALTER, DROP, DELETE, and INSERT) showing in the Lake Formation console.

```
grant all
on external table mySchema.mytable
to iam_role 'arn:aws:iam::123456789012:role/myGrantee';
```

Superusers can access all objects regardless of GRANT and REVOKE commands that set object privileges.

## Usage notes for column-level access control

The following usage notes apply to column-level privileges on Amazon Redshift tables and views. These notes describe tables; the same notes apply to views unless we explicitly note an exception.

- For an Amazon Redshift table, you can grant only the SELECT and UPDATE privileges at the column level. For an Amazon Redshift view, you can grant only the SELECT privilege at the column level.
- The ALL keyword is a synonym for SELECT and UPDATE privileges combined when used in the context of a column-level GRANT on a table.
- If you don't have the SELECT privilege on all columns in a table, performing a `SELECT *` operation returns only those columns that you have access to.
- `SELECT *` doesn't expand to only accessible columns in the following cases:
  - You can't create a regular view with only accessible columns using `SELECT *`.
  - You can't create a materialized view with only accessible columns using `SELECT *`.
- If you have SELECT or UPDATE privilege on a table or view and add a column, you still have the same privileges on the table or view and thus all its columns.
- Only a table's owner or a superuser can grant column-level privileges.
- The WITH GRANT OPTION clause isn't supported for column-level privileges.
- You can't hold the same privilege at both the table level and the column level. For example, the user `data_scientist` can't have both SELECT privilege on the table `employee` and SELECT privilege on the column `employee.department`. Consider the following results when granting the same privilege to a table and a column within the table:

- If a user has a table-level privilege on a table, then granting the same privilege at the column level has no effect.
- If a user has a table-level privilege on a table, then revoking the same privilege for one or more columns of the table returns an error. Instead, revoke the privilege at the table level.
- If a user has a column-level privilege, then granting the same privilege at the table level returns an error.
- If a user has a column-level privilege, then revoking the same privilege at the table level revokes both column and table privileges for all columns on the table.
- You can't grant column-level privileges on late-binding views.
- To create a materialized view, you must have table-level SELECT privilege on the base tables. Even if you have column-level privileges on specific columns, you can't create a materialized view on only those columns. However, you can grant SELECT privilege to columns of a materialized view, similar to regular views.
- To look up grants of column-level privileges, use the [PG\\_ATTRIBUTE\\_INFO \(p. 1671\)](#) view.

## Usage notes for granting the ASSUMEROLE permission

The following usage notes apply to granting the ASSUMEROLE permission in Amazon Redshift.

You use the ASSUMEROLE permission to control IAM role access permissions for database users, roles, or groups on commands such as COPY, UNLOAD, EXTERNAL FUNCTION, or CREATE MODEL. After you grant the ASSUMEROLE permission to a user, role, or group for an IAM role, the user, role, or group can assume that role when running the command. The ASSUMEROLE permission enables you to grant access to the appropriate commands as required.

Only a database superuser can grant or revoke the ASSUMEROLE permission for users, roles, and groups. A superuser always retains the ASSUMEROLE permission.

To enable the use of the ASSUMEROLE permission for users, roles, and groups, a superuser performs the following two actions:

- Run the following statement once on the cluster:

```
revoke assumeroles on all from public for all;
```

- Grant the ASSUMEROLE permission to users, roles, and groups for the appropriate commands.

You can specify role chaining in the ON clause when granting the ASSUMEROLE permission. You use commas to separate roles in a role chain, for example, Role1,Role2,Role3. If role chaining was specified when granting the ASSUMEROLE permission, you must specify the role chain when performing operations granted by the ASSUMEROLE permission. You can't specify individual roles within the role chain when performing operations granted by the ASSUMEROLE permission. For example, if a user, role, or group is granted the role chain Role1,Role2,Role3, you can't specify only Role1 to perform operations.

If a user attempts to perform a COPY, UNLOAD, EXTERNAL FUNCTION, or CREATE MODEL operation and hasn't been granted the ASSUMEROLE permission, a message similar to the following appears.

```
ERROR: User awsuser does not have ASSUMEROLE permission on IAM role
"arn:aws:iam::123456789012:role/RoleA" for COPY
```

To list users that have been granted access to IAM roles and commands through the ASSUMEROLE permission, see [HAS\\_ASSUMEROLE\\_PRIVILEGE \(p. 1391\)](#). To list IAM roles and command permissions that have been granted to a user that you specify, see [PG\\_GET\\_IAM\\_ROLE\\_BY\\_USER \(p. 1397\)](#).

To list users, roles, and groups that have been granted access to an IAM role that you specify, see [PG\\_GET\\_GRANTEE\\_BY\\_IAM\\_ROLE \(p. 1396\)](#).

## Usage notes for granting machine learning permissions

You can't directly grant or revoke permissions related to an ML function. An ML function belongs to an ML model and permissions are controlled through the model. Instead, you can grant permissions related to the ML model. The following example demonstrates how to grant permissions to all users to run the ML function associated with the model `customer_churn`.

```
GRANT EXECUTE ON MODEL customer_churn TO PUBLIC;
```

You can also grant all permissions to a user for the ML model `customer_churn`.

```
GRANT ALL on MODEL customer_churn TO ml_user;
```

Granting the EXECUTE permission related to an ML function will fail if there is an ML function in the schema, even if that ML function already has the EXECUTE permission through GRANT EXECUTE ON MODEL. We recommend using a separate schema when using the CREATE MODEL command to keep the ML functions in a separate schema by themselves. The following example demonstrates how to do so.

```
CREATE MODEL ml_schema.customer_churn
FROM customer_data
TARGET churn
FUNCTION ml_schema.customer_churn_prediction
IAM_ROLE default
SETTINGS (
 S3_BUCKET 'your-s3-bucket'
);
```

## Examples

The following example grants the SELECT privilege on the SALES table to the user fred.

```
grant select on table sales to fred;
```

The following example grants the SELECT privilege on all tables in the QA\_TICKIT schema to the user fred.

```
grant select on all tables in schema qa_tickit to fred;
```

The following example grants all schema privileges on the schema QA\_TICKIT to the user group QA\_USERS. Schema privileges are CREATE and USAGE. USAGE grants users access to the objects in the schema, but doesn't grant privileges such as INSERT or SELECT on those objects. Grant privileges on each object separately.

```
create group qa_users;
grant all on schema qa_tickit to group qa_users;
```

The following example grants all privileges on the SALES table in the QA\_TICKIT schema to all users in the group QA\_USERS.

```
grant all on table qa_tickit.sales to group qa_users;
```

The following example grants all privileges on the SALES table in the QA\_TICKIT schema to all users in the groups QA\_USERS and RO\_USERS.

```
grant all on table qa_tickit.sales to group qa_users, group ro_users;
```

The following example grants the DROP privilege on the SALES table in the QA\_TICKIT schema to all users in the group QA\_USERS.

```
grant drop on table qa_tickit.sales to group qa_users;>
```

The following sequence of commands shows how access to a schema doesn't grant privileges on a table in the schema.

```
create user schema_user in group qa_users password 'Abcd1234';
create schema qa_tickit;
create table qa_tickit.test (col1 int);
grant all on schema qa_tickit to schema_user;

set session authorization schema_user;
select current_user;

current_user

schema_user
(1 row)

select count(*) from qa_tickit.test;

ERROR: permission denied for relation test [SQL State=42501]

set session authorization dw_user;
grant select on table qa_tickit.test to schema_user;
set session authorization schema_user;
select count(*) from qa_tickit.test;

count

0
(1 row)
```

The following sequence of commands shows how access to a view doesn't imply access to its underlying tables. The user called VIEW\_USER can't select from the DATE table, although this user has been granted all privileges on VIEW\_DATE.

```
create user view_user password 'Abcd1234';
create view view_date as select * from date;
grant all on view_date to view_user;
set session authorization view_user;
select current_user;

current_user

view_user
(1 row)
```

```
select count(*) from view_date;

count

365
(1 row)

select count(*) from date;

ERROR: permission denied for relation date
```

The following example grants SELECT privilege on the `cust_name` and `cust_phone` columns of the `cust_profile` table to the user `user1`.

```
grant select(cust_name, cust_phone) on cust_profile to user1;
```

The following example grants SELECT privilege on the `cust_name` and `cust_phone` columns and UPDATE privilege on the `cust_contact_preference` column of the `cust_profile` table to the `sales_group` group.

```
grant select(cust_name, cust_phone), update(cust_contact_preference) on cust_profile to
group sales_group;
```

The following example shows the usage of the ALL keyword to grant both SELECT and UPDATE privileges on three columns of the table `cust_profile` to the `sales_admin` group.

```
grant ALL(cust_name, cust_phone,cust_contact_preference) on cust_profile to group
sales_admin;
```

The following example grants the SELECT privilege on the `cust_name` column of the `cust_profile_vw` view to the `user2` user.

```
grant select(cust_name) on cust_profile_vw to user2;
```

## Examples of granting the USAGE privilege for datashares

The following examples show GRANT datasharing usage permissions on a specific database or schema created from a datashare. This USAGE permission doesn't grant usage permission to databases that aren't created from the specified datashare. You can only GRANT or REVOKE ALTER or SHARE permissions on a datashare to users and user groups.

The following example grants the USAGE privilege on the `salesshare` datashare to the specified namespace.

```
GRANT USAGE ON DATASHARE salesshare TO NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

The following example grants the USAGE privilege on the `sales_db` to Bob.

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

The following example GRANT USAGE privilege on the `sales_schema` to the `Analyst_group`.

```
GRANT USAGE ON SCHEMA sales_schema TO GROUP Analyst_group;
```

## Examples of granting the ASSUMEROLE privilege

The following are examples of granting the ASSUMEROLE privilege.

The following example shows the REVOKE statement that a superuser runs once on the cluster to enable the use of the ASSUMEROLE privilege for users and groups. Then, the superuser grants the ASSUMEROLE privilege to users and groups for the appropriate commands. For information on enabling the use of the ASSUMEROLE privilege for users and groups, see [Usage notes for granting the ASSUMEROLE permission \(p. 897\)](#).

```
revoke assumeroles on all from public for all;
```

The following example grants the ASSUMEROLE privilege to the user `reg_user1` for the IAM role `Redshift-S3-Read` to perform COPY operations.

```
grant assumeroles on 'arn:aws:iam::123456789012:role/Redshift-S3-Read'
to reg_user1 for copy;
```

The following example grants the ASSUMEROLE privilege to the user `reg_user1` for the IAM role chain `RoleA, RoleB` to perform UNLOAD operations.

```
grant assumeroles
on 'arn:aws:iam::123456789012:role/RoleA,arn:aws:iam::210987654321:role/RoleB'
to reg_user1
for unload;
```

The following is an example of the UNLOAD command using the IAM role chain `RoleA, RoleB`.

```
unload ('select * from venue limit 10')
to 's3://companyb/redshift/venue_pipe'_
iam_role 'arn:aws:iam::123456789012:role/RoleA,arn:aws:iam::210987654321:role/RoleB';
```

The following example grants the ASSUMEROLE privilege to the user `reg_user1` for the IAM role `Redshift-Exfunc` to create external functions.

```
grant assumeroles on 'arn:aws:iam::123456789012:role/Redshift-Exfunc'
to reg_user1 for external function;
```

The following example grants the ASSUMEROLE privilege to the user `reg_user1` for the IAM role `Redshift-Model` to create machine learning models.

```
grant assumeroles on 'arn:aws:iam::123456789012:role/Redshift-ML'
to reg_user1 for create model;
```

## Examples of granting the ROLE privileges

The following example grants `sample_role1` to `user1`.

```
CREATE ROLE sample_role1;
GRANT ROLE sample_role1 TO user1;
```

The following example grants sample\_role1 to user1 with the WITH ADMIN OPTION, sets the current session for user1, and user1 grants sample\_role1 to user2.

```
GRANT ROLE sample_role1 TO user1 WITH ADMIN OPTION;
SET SESSION AUTHORIZATION user1;
GRANT ROLE sample_role1 TO user2;
```

The following example grants sample\_role1 to sample\_role2.

```
GRANT ROLE sample_role1 TO ROLE sample_role2;
```

The following example grants sample\_role2 to sample\_role3 and sample\_role4. Then it attempts to grants sample\_role3 to sample\_role1.

```
GRANT ROLE sample_role2 TO ROLE sample_role3;
GRANT ROLE sample_role3 TO ROLE sample_role2;
ERROR: cannot grant this role, a circular dependency was detected between these roles
```

The following example grants the CREATE USER system privileges to sample\_role1.

```
GRANT CREATE USER TO ROLE sample_role1;
```

The following example grants the system-defined role sys:dba to user1.

```
GRANT ROLE sys:dba TO user1;
```

The following example attempts to grant sample\_role3 in a circular dependency to sample\_role2.

```
CREATE ROLE sample_role3;
GRANT ROLE sample_role2 TO ROLE sample_role3;
GRANT ROLE sample_role3 TO ROLE sample_role2; -- fail
ERROR: cannot grant this role, a circular dependency was detected between these roles
```

## INSERT

### Topics

- [Syntax \(p. 903\)](#)
- [Parameters \(p. 903\)](#)
- [Usage notes \(p. 904\)](#)
- [INSERT examples \(p. 904\)](#)

Inserts new rows into a table. You can insert a single row with the VALUES syntax, multiple rows with the VALUES syntax, or one or more rows defined by the results of a query (INSERT INTO...SELECT).

#### Note

We strongly encourage you to use the [COPY \(p. 677\)](#) command to load large amounts of data. Using individual INSERT statements to populate a table might be prohibitively slow. Alternatively, if your data already exists in other Amazon Redshift database tables, use INSERT INTO SELECT or [CREATE TABLE AS \(p. 838\)](#) to improve performance. For more information about using the COPY command to load tables, see [Loading data \(p. 63\)](#).

#### Note

The maximum size for a single SQL statement is 16 MB.

## Syntax

```
INSERT INTO table_name [(column [, ...])]
{DEFAULT VALUES |
VALUES ({ expression | DEFAULT } [, ...])
[, ({ expression | DEFAULT } [, ...])
[, ...]] | |
query }
```

## Parameters

### *table\_name*

A temporary or persistent table. Only the owner of the table or a user with INSERT privilege on the table can insert rows. If you use the *query* clause to insert rows, you must have SELECT privilege on the tables named in the query.

#### **Note**

Use INSERT (external table) to insert results of a SELECT query into existing tables on external catalog. For more information, see [INSERT \(external table\) \(p. 906\)](#).

### *column*

You can insert values into one or more columns of the table. You can list the target column names in any order. If you don't specify a column list, the values to be inserted must correspond to the table columns in the order in which they were declared in the CREATE TABLE statement. If the number of values to be inserted is less than the number of columns in the table, the first *n* columns are loaded.

Either the declared default value or a null value is loaded into any column that isn't listed (implicitly or explicitly) in the INSERT statement.

### DEFAULT VALUES

If the columns in the table were assigned default values when the table was created, use these keywords to insert a row that consists entirely of default values. If any of the columns don't have default values, nulls are inserted into those columns. If any of the columns are declared NOT NULL, the INSERT statement returns an error.

### VALUES

Use this keyword to insert one or more rows, each row consisting of one or more values. The VALUES list for each row must align with the column list. To insert multiple rows, use a comma delimiter between each list of expressions. Do not repeat the VALUES keyword. All VALUES lists for a multiple-row INSERT statement must contain the same number of values.

### *expression*

A single value or an expression that evaluates to a single value. Each value must be compatible with the data type of the column where it is being inserted. If possible, a value whose data type doesn't match the column's declared data type is automatically converted to a compatible data type. For example:

- A decimal value 1.1 is inserted into an INT column as 1.
- A decimal value 100.8976 is inserted into a DEC(5,2) column as 100.90.

You can explicitly convert a value to a compatible data type by including type cast syntax in the expression. For example, if column COL1 in table T1 is a CHAR(3) column:

```
insert into t1(col1) values('Incomplete'::char(3));
```

This statement inserts the value Inc into the column.

For a single-row INSERT VALUES statement, you can use a scalar subquery as an expression. The result of the subquery is inserted into the appropriate column.

**Note**

Subqueries aren't supported as expressions for multiple-row INSERT VALUES statements.

**DEFAULT**

Use this keyword to insert the default value for a column, as defined when the table was created. If no default value exists for a column, a null is inserted. You can't insert a default value into a column that has a NOT NULL constraint if that column doesn't have an explicit default value assigned to it in the CREATE TABLE statement.

*query*

Insert one or more rows into the table by defining any query. All of the rows that the query produces are inserted into the table. The query must return a column list that is compatible with the columns in the table, but the column names don't have to match.

## Usage notes

**Note**

We strongly encourage you to use the [COPY \(p. 677\)](#) command to load large amounts of data. Using individual INSERT statements to populate a table might be prohibitively slow. Alternatively, if your data already exists in other Amazon Redshift database tables, use INSERT INTO SELECT or [CREATE TABLE AS \(p. 838\)](#) to improve performance. For more information about using the COPY command to load tables, see [Loading data \(p. 63\)](#).

The data format for the inserted values must match the data format specified by the CREATE TABLE definition.

After inserting a large number of new rows into a table:

- Vacuum the table to reclaim storage space and re-sort rows.
- Analyze the table to update statistics for the query planner.

When values are inserted into DECIMAL columns and they exceed the specified scale, the loaded values are rounded up as appropriate. For example, when a value of 20.259 is inserted into a DECIMAL(8,2) column, the value that is stored is 20.26.

You can insert into a GENERATED BY DEFAULT AS IDENTITY column. You can update columns defined as GENERATED BY DEFAULT AS IDENTITY with values that you supply. For more information, see [GENERATED BY DEFAULT AS IDENTITY \(p. 826\)](#).

## INSERT examples

The CATEGORY table in the TICKIT database contains the following rows:

| catid | catgroup | catname  | catdesc                         |
|-------|----------|----------|---------------------------------|
| 1     | Sports   | MLB      | Major League Baseball           |
| 2     | Sports   | NHL      | National Hockey League          |
| 3     | Sports   | NFL      | National Football League        |
| 4     | Sports   | NBA      | National Basketball Association |
| 5     | Sports   | MLS      | Major League Soccer             |
| 6     | Shows    | Musicals | Musical theatre                 |
| 7     | Shows    | Plays    | All non-musical theatre         |
| 8     | Shows    | Opera    | All opera and light opera       |
| 9     | Concerts | Pop      | All rock and pop music concerts |

```
10 | Concerts | Jazz | All jazz singers and bands
11 | Concerts | Classical | All symphony, concerto, and choir concerts
(11 rows)
```

Create a `CATEGORY_STAGE` table with a similar schema to the `CATEGORY` table but define default values for the columns:

```
create table category_stage
(catid smallint default 0,
catgroup varchar(10) default 'General',
catname varchar(10) default 'General',
catdesc varchar(50) default 'General');
```

The following `INSERT` statement selects all of the rows from the `CATEGORY` table and inserts them into the `CATEGORY_STAGE` table.

```
insert into category_stage
(select * from category);
```

The parentheses around the query are optional.

This command inserts a new row into the `CATEGORY_STAGE` table with a value specified for each column in order:

```
insert into category_stage values
(12, 'Concerts', 'Comedy', 'All stand-up comedy performances');
```

You can also insert a new row that combines specific values and default values:

```
insert into category_stage values
(13, 'Concerts', 'Other', default);
```

Run the following query to return the inserted rows:

```
select * from category_stage
where catid in(12,13) order by 1;
-----+-----+-----+-----+
catid | catgroup | catname | catdesc
-----+-----+-----+-----+
12 | Concerts | Comedy | All stand-up comedy performances
13 | Concerts | Other | General
(2 rows)
```

The following examples show some multiple-row `INSERT VALUES` statements. The first example inserts specific `CATID` values for two rows and default values for the other columns in both rows.

```
insert into category_stage values
(14, default, default, default),
(15, default, default, default);

select * from category_stage where catid in(14,15) order by 1;
-----+-----+-----+-----+
catid | catgroup | catname | catdesc
-----+-----+-----+-----+
14 | General | General | General
15 | General | General | General
(2 rows)
```

The next example inserts three rows with various combinations of specific and default values:

```
insert into category_stage values
(default, default, default, default),
(20, default, 'Country', default),
(21, 'Concerts', 'Rock', default);

select * from category_stage where catid in(0,20,21) order by 1;
catid | catgroup | catname | catdesc
-----+-----+-----+
0 | General | General | General
20 | General | Country | General
21 | Concerts | Rock | General
(3 rows)
```

The first set of VALUES in this example produces the same results as specifying DEFAULT VALUES for a single-row INSERT statement.

The following examples show INSERT behavior when a table has an IDENTITY column. First, create a new version of the CATEGORY table, then insert rows into it from CATEGORY:

```
create table category_ident
(catid int identity not null,
catgroup varchar(10) default 'General',
catname varchar(10) default 'General',
catdesc varchar(50) default 'General');

insert into category_ident(catgroup,catname,catdesc)
select catgroup,catname,catdesc from category;
```

Note that you can't insert specific integer values into the CATID IDENTITY column. IDENTITY column values are automatically generated.

The following example demonstrates that subqueries can't be used as expressions in multiple-row INSERT VALUES statements:

```
insert into category(catid) values
((select max(catid)+1 from category)),
((select max(catid)+2 from category));

ERROR: can't use subqueries in multi-row VALUES
```

## INSERT (external table)

Inserts the results of a SELECT query into existing external tables on external catalog such as for AWS Glue, AWS Lake Formation, or an Apache Hive metastore. Use the same AWS Identity and Access Management (IAM) role used for the CREATE EXTERNAL SCHEMA command to interact with external catalogs and Amazon S3.

For nonpartitioned tables, the INSERT (external table) command writes data to the Amazon S3 location defined in the table, based on the specified table properties and file format.

For partitioned tables, INSERT (external table) writes data to the Amazon S3 location according to the partition key specified in the table. It also automatically registers new partitions in the external catalog after the INSERT operation completes.

You can't run INSERT (external table) within a transaction block (BEGIN ... END). For more information about transactions, see [Serializable isolation \(p. 119\)](#).

## Syntax

```
INSERT INTO external_schema.table_name
{ select_statement }
```

## Parameters

*external\_schema.table\_name*

The name of an existing external schema and a target external table to insert into.

*select\_statement*

A statement that inserts one or more rows into the external table by defining any query. All of the rows that the query produces are written to Amazon S3 in either text or Parquet format based on the table definition. The query must return a column list that is compatible with the column data types in the external table. However, the column names don't have to match.

## Usage notes

The number of columns in the SELECT query must be the same as the sum of data columns and partition columns. The location and the data type of each data column must match that of the external table. The location of partition columns must be at the end of the SELECT query, in the same order they were defined in CREATE EXTERNAL TABLE command. The column names don't have to match.

In some cases, you might want to run the INSERT (external table) command on an AWS Glue Data Catalog or a Hive metastore. In the case of AWS Glue, the IAM role used to create the external schema must have both read and write permissions on Amazon S3 and AWS Glue. If you use an AWS Lake Formation catalog, this IAM role becomes the owner of the new Lake Formation table. This IAM role must at least have the following permissions:

- SELECT, INSERT, UPDATE permission on the external table
- Data location permission on the Amazon S3 path of the external table

To ensure that file names are unique, Amazon Redshift uses the following format for the name of each file uploaded to Amazon S3 by default.

*<date>\_<time>\_<microseconds>\_<query\_id>\_<slice-number>\_part\_<part-number>.<format>*.

An example is 20200303\_004509\_810669\_1007\_0001\_part\_00.parquet.

Consider the following when running the INSERT (external table) command:

- External tables that have a format other than PARQUET or TEXTFILE aren't supported.
- This command supports existing table properties such as 'write.parallel', 'write.maxfilesize.mb', 'compression\_type', and 'serialization.null.format'. To update those values, run the ALTER TABLE SET TABLE PROPERTIES command.
- The 'numRows' table property is automatically updated toward the end of the INSERT operation. The table property must be defined or added to the table already if it wasn't created by CREATE EXTERNAL TABLE AS operation.
- The LIMIT clause isn't supported in the outer SELECT query. Instead, use a nested LIMIT clause.
- You can use the [STL\\_UNLOAD\\_LOG \(p. 1564\)](#) table to track the files that got written to Amazon S3 by each INSERT (external table) operation.
- Amazon Redshift supports only Amazon S3 standard encryption for INSERT (external table).

## INSERT (external table) examples

The following example inserts the results of the SELECT statement into the external table.

```
INSERT INTO spectrum.lineitem
SELECT * FROM local_lineitem;
```

The following example inserts the results of the SELECT statement into a partitioned external table using static partitioning. The partition columns are hardcoded in the SELECT statement. The partition columns must be at the end of the query.

```
INSERT INTO spectrum.customer
SELECT name, age, gender, 'May', 28 FROM local_customer;
```

The following example inserts the results of the SELECT statement into a partitioned external table using dynamic partitioning. The partition columns aren't hardcoded. Data is automatically added to the existing partition folders, or to new folders if a new partition is added.

```
INSERT INTO spectrum.customer
SELECT name, age, gender, month, day FROM local_customer;
```

## LOCK

Restricts access to a database table. This command is only meaningful when it is run inside a transaction block.

The LOCK command obtains a table-level lock in "ACCESS EXCLUSIVE" mode, waiting if necessary for any conflicting locks to be released. Explicitly locking a table in this way causes reads and writes on the table to wait when they are attempted from other transactions or sessions. An explicit table lock created by one user temporarily prevents another user from selecting data from that table or loading data into it. The lock is released when the transaction that contains the LOCK command completes.

Less restrictive table locks are acquired implicitly by commands that refer to tables, such as write operations. For example, if a user tries to read data from a table while another user is updating the table, the data that is read will be a snapshot of the data that has already been committed. (In some cases, queries will stop if they violate serializable isolation rules.) See [Managing concurrent write operations \(p. 118\)](#).

Some DDL operations, such as DROP TABLE and TRUNCATE, create exclusive locks. These operations prevent data reads.

If a lock conflict occurs, Amazon Redshift displays an error message to alert the user who started the transaction in conflict. The transaction that received the lock conflict is stopped. Every time a lock conflict occurs, Amazon Redshift writes an entry to the [STL\\_TR\\_CONFLICT \(p. 1561\)](#) table.

## Syntax

```
LOCK [TABLE] table_name [, ...]
```

## Parameters

### TABLE

Optional keyword.

*table\_name*

Name of the table to lock. You can lock more than one table by using a comma-delimited list of table names. You can't lock views.

## Example

```
begin;

lock event, sales;

...
```

## MERGE

Conditionally merges rows from a source table into a target table. Traditionally, this can only be achieved by using multiple insert, update or delete statements separately. For more information on the operations that MERGE lets you combine, see [UPDATE](#), [DELETE](#), and [INSERT](#).

## Syntax

```
MERGE INTO target_table
USING source_table [[AS] alias] ON match_condition
WHEN MATCHED THEN { UPDATE SET col_name = { expr } [,...] | DELETE }
WHEN NOT MATCHED THEN INSERT [(col_name [,...])] VALUES ({ expr } [, ...])
```

## Parameters

*target\_table*

The temporary or permanent table that the MERGE statement merges into.

*source\_table*

The temporary or permanent table supplying the rows to merge into *target\_table*. *source\_table* can also be a Spectrum table, Postgres federated table, or MySQL federated table. *source\_table* can't be a view or a subquery.

*alias*

The temporary alternative name for *source\_table*.

This parameter is optional. Preceding *alias* with AS is also optional.

*match\_condition*

Specifies equal predicates between the source table column and target table column that are used to determine whether the rows in *source\_table* can be matched with rows in *target\_table*. If the condition is met, MERGE runs *matched\_clause* for that row. Otherwise MERGE runs *not\_matched\_clause* for that row.

**WHEN MATCHED**

Specifies the action to be executed when the match condition between a source row and a target row evaluates to True. You can specify either an UPDATE action or a DELETE action.

**UPDATE**

Updates the matched row in *target\_table*. Only values in the *col\_name* you specify are updated.

#### DELETE

Deletes the matched row in *target\_table*.

#### WHEN NOT MATCHED

Specifies the action to be executed when the match condition is evaluated to False or Unknown. You can only specify the INSERT insert action for this clause.

#### INSERT

Inserts one row into *target\_table*. The target *col\_name* can be listed in any order. If you don't provide any *col\_name* values, the default order is all the table's columns in their declared order.

#### *col\_name*

One or more column names that you want to modify. Don't include the table name when specifying the target column.

#### *expr*

The expression defining the new value for *col\_name*.

## Usage notes

- To run MERGE statements, you must be the owner of both *source\_table* and *target\_table*, or have the SELECT permission for those tables. Additionally, you must have UPDATE, DELETE, and INSERT permissions for *target\_table* depending on the operations included in your MERGE statement.
- *target\_table* can't be a system table, catalog table, or external table.
- *source\_table* and *target\_table* can't be the same table.
- You can't use the WITH clause in a MERGE statement.
- Rows in *target\_table* can't match multiple rows in *source\_table*.

Consider the following example:

```

CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));

INSERT INTO target VALUES (1, 'Bob'), (2, 'John');
INSERT INTO source VALUES (1, 'Tony'), (1, 'Alice'), (3, 'Bill');

MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN UPDATE SET id = source.id, name = source.name
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);
ERROR: Found multiple matches to update the same tuple.

MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN DELETE
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);
ERROR: Found multiple matches to update the same tuple.

```

In both MERGE statements, the operation fails because there are multiple rows in the source table with an ID value of 1.

- *match\_condition* and *expr* can't partially reference SUPER type columns. For example, if your SUPER type object is an array or a structure, you can't use individual elements of that column for *match\_condition* or *expr*, but you can use the entire column.

Consider the following example:

```

CREATE TABLE IF NOT EXISTS target (key INT, value SUPER);

```

```

CREATE TABLE IF NOT EXISTS source (key INT, value SUPER);

INSERT INTO target VALUES (1, JSON_PARSE('{"key": 88}'));
INSERT INTO source VALUES (1, ARRAY(1, 'John')), (2, ARRAY(2, 'Bill'));

MERGE INTO target USING source ON target.key = source.key
WHEN matched THEN UPDATE SET value = source.value[0]
WHEN NOT matched THEN INSERT VALUES (source.key, source.value[0]);
ERROR: Partial reference of SUPER column is not supported in MERGE statement.

```

For more information on the SUPER type, see [SUPER type](#).

- If *source\_table* is large, defining the join columns from both *target\_table* and *source\_table* as the distribution keys can improve performance.

## Examples

The following example creates two tables, then runs a MERGE operation on them, updating matching rows in the target table and inserting rows that don't match. Then it inserts another value into the source table and runs another MERGE operation, this time deleting matching rows and inserting the new row from the source table.

First create and populate the source and target tables.

```

CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));

INSERT INTO target VALUES (101, 'Bob'), (102, 'John'), (103, 'Susan');
INSERT INTO source VALUES (102, 'Tony'), (103, 'Alice'), (104, 'Bill');

SELECT * FROM target;
id | name
---+-----
101 | Bob
102 | John
103 | Susan
(3 rows)

SELECT * FROM source;
id | name
---+-----
102 | Tony
103 | Alice
104 | Bill
(3 rows)

```

Next, merge the source table into the target table, updating the target table with matching rows and insert rows from the source table that have no match.

```

MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN UPDATE SET id = source.id, name = source.name
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);

SELECT * FROM target;
id | name
---+-----
101 | Bob
102 | Tony
103 | Alice
104 | Bill
(4 rows)

```

Note that the rows with id values of 102 and 103 are updated to match the name values from the target table. Also, a new row with an id value of 104 and name value of Bill is inserted into the target table.

Next, insert a new row into the source table.

```
INSERT INTO source VALUES (105, 'David');

SELECT * FROM source;
+-----+-----+
| id | name |
+-----+-----+
102	Tony
103	Alice
104	Bill
105	David
+-----+
(4 rows)
```

Finally, run a merge operation deleting matching rows in the target table, and inserting rows that don't match.

```
MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN DELETE
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);

SELECT * FROM target;
+-----+-----+
| id | name |
+-----+-----+
| 101 | Bob |
| 105 | David |
+-----+
(2 rows)
```

The rows with id values 102, 103, and 104 are deleted from the target table, and a new row with an id value of 105 and name value of David is inserted into the target table.

## See also

[INSERT \(p. 902\)](#), [UPDATE \(p. 1010\)](#), [DELETE \(p. 858\)](#)

## PREPARE

Prepare a statement for execution.

PREPARE creates a prepared statement. When the PREPARE statement is run, the specified statement (SELECT, INSERT, UPDATE, or DELETE) is parsed, rewritten, and planned. When an EXECUTE command is then issued for the prepared statement, Amazon Redshift may optionally revise the query execution plan (to improve performance based on the specified parameter values) before running the prepared statement.

## Syntax

```
PREPARE plan_name [(datatype [, ...])] AS statement
```

## Parameters

*plan\_name*

An arbitrary name given to this particular prepared statement. It must be unique within a single session and is subsequently used to run or deallocate a previously prepared statement.

**datatype**

The data type of a parameter to the prepared statement. To refer to the parameters in the prepared statement itself, use \$1, \$2, and so on.

**statement**

Any SELECT, INSERT, UPDATE, or DELETE statement.

## Usage notes

Prepared statements can take parameters: values that are substituted into the statement when it is run. To include parameters in a prepared statement, supply a list of data types in the PREPARE statement, and, in the statement to be prepared itself, refer to the parameters by position using the notation \$1, \$2, ... When running the statement, specify the actual values for these parameters in the EXECUTE statement. For more details, see [EXECUTE \(p. 880\)](#).

Prepared statements only last for the duration of the current session. When the session ends, the prepared statement is discarded, so it must be re-created before being used again. This also means that a single prepared statement can't be used by multiple simultaneous database clients; however, each client can create its own prepared statement to use. The prepared statement can be manually removed using the DEALLOCATE command.

Prepared statements have the largest performance advantage when a single session is being used to run a large number of similar statements. As mentioned, for each new execution of a prepared statement, Amazon Redshift may revise the query execution plan to improve performance based on the specified parameter values. To examine the query execution plan that Amazon Redshift has chosen for any specific EXECUTE statements, use the [EXPLAIN \(p. 881\)](#) command.

For more information on query planning and the statistics collected by Amazon Redshift for query optimization, see the [ANALYZE \(p. 663\)](#) command.

## Examples

Create a temporary table, prepare INSERT statement and then run it:

```
DROP TABLE IF EXISTS prep1;
CREATE TABLE prep1 (c1 int, c2 char(20));
PREPARE prep_insert_plan (int, char)
AS insert into prep1 values ($1, $2);
EXECUTE prep_insert_plan (1, 'one');
EXECUTE prep_insert_plan (2, 'two');
EXECUTE prep_insert_plan (3, 'three');
DEALLOCATE prep_insert_plan;
```

Prepare a SELECT statement and then run it:

```
PREPARE prep_select_plan (int)
AS select * from prep1 where c1 = $1;
EXECUTE prep_select_plan (2);
EXECUTE prep_select_plan (3);
DEALLOCATE prep_select_plan;
```

## See also

[DEALLOCATE \(p. 854\)](#), [EXECUTE \(p. 880\)](#)

## REFRESH MATERIALIZED VIEW

Refreshes a materialized view.

When you create a materialized view, its contents reflect the state of the underlying database table or tables at that time. The data in the materialized view remains unchanged, even when applications make changes to the data in the underlying tables. To update the data in a materialized view, you can use the REFRESH MATERIALIZED VIEW statement at any time. When you use this statement, Amazon Redshift identifies changes that have taken place in the base table or tables, and then applies those changes to the materialized view.

For more information about materialized views, see [Creating materialized views in Amazon Redshift \(p. 198\)](#).

### Syntax

```
REFRESH MATERIALIZED VIEW mv_name
```

### Parameters

*mv\_name*

The name of the materialized view to be refreshed.

### Usage notes

Only the owner of a materialized view can perform a REFRESH MATERIALIZED VIEW operation on that materialized view. Furthermore, the owner must have SELECT privilege on the underlying base tables to successfully run REFRESH MATERIALIZED VIEW.

The REFRESH MATERIALIZED VIEW command runs as a transaction of its own. Amazon Redshift transaction semantics are followed to determine what data from base tables is visible to the REFRESH command, or when the changes made by the REFRESH command are made visible to other transactions running in Amazon Redshift.

- For incremental materialized views, REFRESH MATERIALIZED VIEW uses only those base table rows that are already committed. Therefore, if the refresh operation runs after a data manipulation language (DML) statement in the same transaction, then changes of that DML statement aren't visible to refresh.
- For a full refresh of a materialized view, REFRESH MATERIALIZED VIEW sees all base table rows visible to the refresh transaction, according to usual Amazon Redshift transaction semantics.
- Depending on the input argument type, Amazon Redshift still supports incremental refresh for materialized views for the following functions with specific input argument types: DATE (timestamp), DATE\_PART (date, time, interval, time-tz), DATE\_TRUNC (timestamp, interval).

Some operations in Amazon Redshift interact with materialized views. Some of these operations might force a REFRESH MATERIALIZED VIEW operation to fully recompute the materialized view even though the query defining the materialized view only uses the SQL features eligible for incremental refresh. For example:

- Background vacuum operations might be blocked if materialized views aren't refreshed. After an internally defined threshold period, a vacuum operation is allowed to run. When this vacuum operation happens, any dependent materialized views are marked for recomputation upon the next

refresh (even if they are incremental). For information about VACUUM, see [VACUUM \(p. 1015\)](#). For more information about events and state changes, see [STL\\_MV\\_STATE \(p. 1532\)](#).

- Some user-initiated operations on base tables force a materialized view to be fully recomputed next time that a REFRESH operation is run. Examples of such operations are a manually invoked VACUUM, a classic resize, an ALTER DISTKEY operation, an ALTER SORTKEY operation, and a truncate operation. For more information about events and state changes, see [STL\\_MV\\_STATE \(p. 1532\)](#).

## Limitations for incremental refresh

Amazon Redshift currently doesn't support incremental refresh for materialized views that are defined with a query using any of the following SQL elements:

- OUTER JOIN (RIGHT, LEFT, or FULL).
- Set operations: UNION, INTERSECT, EXCEPT, MINUS.
- UNION ALL when it occurs in a subquery and an aggregate function or a GROUP BY clause is present in the query.
- Aggregate functions: MEDIAN, PERCENTILE\_CONT, LISTAGG, STDDEV\_SAMP, STDDEV\_POP, APPROXIMATE COUNT, APPROXIMATE PERCENTILE, and bitwise aggregate functions.

### Note

The COUNT, SUM, MIN, MAX, and AVG aggregate functions are supported.

- DISTINCT aggregate functions, such as DISTINCT COUNT, DISTINCT SUM, and so on.
- Window functions.
- A query that uses temporary tables for query optimization, such as optimizing common subexpressions.
- Subqueries in any place other than the FROM clause.
- External tables referenced as base tables in the query that defines the materialized view.
- Mutable functions, such as date-time functions, RANDOM and non-STABLE user-defined functions.

## Examples

The following example refreshes the `tickets_mv` materialized view.

```
REFRESH MATERIALIZED VIEW tickets_mv;
```

## RESET

Restores the value of a configuration parameter to its default value.

You can reset either a single specified parameter or all parameters at once. To set a parameter to a specific value, use the [SET \(p. 977\)](#) command. To display the current value of a parameter, use the [SHOW \(p. 981\)](#) command.

## Syntax

```
RESET { parameter_name | ALL }
```

The following statement sets the value of a session context variable to NULL.

```
RESET { variable_name | ALL }
```

## Parameters

*parameter\_name*

Name of the parameter to reset. See [Modifying the server configuration \(p. 1685\)](#) for more documentation about parameters.

**ALL**

Resets all runtime parameters, including all the session context variables.

*variable*

The name of the variable to reset. If the value to RESET is a session context variable, Amazon Redshift sets it to NULL.

## Examples

The following example resets the query\_group parameter to its default value:

```
reset query_group;
```

The following example resets all runtime parameters to their default values.

```
reset all;
```

The following example resets the context variable.

```
RESET app_context.user_id;
```

## REVOKE

Removes access permissions, such as permissions to create, drop, or update tables, from a user or user group.

You can only GRANT or REVOKE USAGE permissions on an external schema to database users and user groups using the ON SCHEMA syntax. When using ON EXTERNAL SCHEMA with AWS Lake Formation, you can only GRANT and REVOKE permissions to an AWS Identity and Access Management (IAM) role. For the list of permissions, see the syntax.

For stored procedures, USAGE ON LANGUAGE plpgsql permissions are granted to PUBLIC by default. EXECUTE ON PROCEDURE permission is granted only to the owner and superusers by default.

Specify in the REVOKE command the permissions that you want to remove. To give permissions, use the [GRANT \(p. 887\)](#) command.

## Syntax

```
REVOKE [GRANT OPTION FOR]
{ { SELECT | INSERT | UPDATE | DELETE | DROP | REFERENCES } [,...] | ALL [PRIVILEGES] }
ON { [TABLE] table_name [, ...] | ALL TABLES IN SCHEMA schema_name [, ...] }
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[RESTRICT]

REVOKE [GRANT OPTION FOR]
{ { CREATE | TEMPORARY | TEMP } [,...] | ALL [PRIVILEGES] }
```

```

ON DATABASE db_name [, ...]
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[RESTRICT]

REVOKE [GRANT OPTION FOR]
{ { CREATE | USAGE } [, ...] | ALL [PRIVILEGES] }
ON SCHEMA schema_name [, ...]
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[RESTRICT]

REVOKE [GRANT OPTION FOR]
EXECUTE
 ON FUNCTION function_name ([[argname] argtype [, ...]]) [, ...]
 FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[RESTRICT]

REVOKE [GRANT OPTION FOR]
{ { EXECUTE } [, ...] | ALL [PRIVILEGES] }
 ON PROCEDURE procedure_name ([[argname] argtype [, ...]]) [, ...]
 FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[RESTRICT]

REVOKE [GRANT OPTION FOR]
USAGE
 ON LANGUAGE language_name [, ...]
 FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[RESTRICT]

```

### Revoking column-level permissions for tables

The following is the syntax for column-level permissions on Amazon Redshift tables and views.

```

REVOKE { { SELECT | UPDATE } (column_name [, ...]) [, ...] | ALL [PRIVILEGES]
(column_name [, ...]) }
 ON { [TABLE] table_name [, ...] }
 FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[RESTRICT]

```

### Revoking ASSUMEROLE permissions

The following is the syntax to revoke the ASSUMEROLE permission from users and groups with a specified role.

```

REVOKE ASSUMEROLE
 ON { 'iam_role' [, ...] | default | ALL }
 FROM { user_name | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
 FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL }

```

### Revoking permissions for Redshift Spectrum for Lake Formation

The following is the syntax for Redshift Spectrum integration with Lake Formation.

```

REVOKE [GRANT OPTION FOR]
{ SELECT | ALL [PRIVILEGES] } (column_list)
 ON EXTERNAL TABLE schema_name.table_name
 FROM { IAM_ROLE iam_role } [, ...]

```

```

REVOKE [GRANT OPTION FOR]
{ { SELECT | ALTER | DROP | DELETE | INSERT } [, ...] | ALL [PRIVILEGES] }
ON EXTERNAL TABLE schema_name.table_name [, ...]
FROM { { IAM_ROLE iam_role } [, ...] | PUBLIC }

REVOKE [GRANT OPTION FOR]
{ { CREATE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
ON EXTERNAL SCHEMA schema_name [, ...]
FROM { IAM_ROLE iam_role } [, ...]

```

### Revoking datashare permissions

The following is the syntax for using REVOKE for datashare permissions for Amazon Redshift.

```

REVOKE { ALTER | SHARE } ON DATASHARE datashare_name
FROM { username [WITH GRANT OPTION] | ROLE role_name | GROUP group_name | PUBLIC }
[, ...]

```

The following is the syntax for using REVOKE for datashare usage permissions for Amazon Redshift.

```

REVOKE USAGE
ON DATASHARE datashare_name
FROM NAMESPACE 'namespaceGUID' [, ...] | ACCOUNT 'accountnumber' [VIA DATA CATALOG]
[, ...]

```

The following is an example of how to revoke usage of a datashare from a Lake Formation account.

```
REVOKE USAGE ON salesshare FROM ACCOUNT '123456789012' VIA DATA CATALOG;
```

The following is the REVOKE syntax for data-sharing usage permissions on a specific database or schema created from a datashare. This USAGE permission doesn't revoke usage permission to databases that are not created from the specified datashare. You can only REVOKE, ALTER, or SHARE permissions on a datashare to users and user groups.

```

REVOKE USAGE ON { DATABASE shared_database_name [, ...] | SCHEMA shared_schema}
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]

```

### Revoking machine learning permissions

The following is the syntax for machine learning model permissions on Amazon Redshift.

```

REVOKE [GRANT OPTION FOR]
CREATE MODEL FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[RESTRICT]

REVOKE [GRANT OPTION FOR]
{ EXECUTE | ALL [PRIVILEGES] }
ON MODEL model_name [, ...]

FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[RESTRICT]

```

### Revoking role permissions

The following is the syntax for revoking role permissions on Amazon Redshift.

```
REVOKE [ADMIN OPTION FOR] { ROLE role_name } [, ...] FROM { user_name } [, ...]
```

```
REVOKE { ROLE role_name } [, ...] FROM { ROLE role_name } [, ...]
```

The following is the syntax for revoking system permissions to roles on Amazon Redshift.

```
REVOKE
{
 { CREATE USER | DROP USER | ALTER USER |
 CREATE SCHEMA | DROP SCHEMA |
 ALTER DEFAULT PRIVILEGES |
 CREATE TABLE | DROP TABLE | ALTER TABLE |
 CREATE OR REPLACE FUNCTION | CREATE OR REPLACE EXTERNAL FUNCTION |
 DROP FUNCTION |
 CREATE OR REPLACE PROCEDURE | DROP PROCEDURE |
 CREATE OR REPLACE VIEW | DROP VIEW |
 CREATE MODEL | DROP MODEL |
 CREATE DATASHARE | ALTER DATASHARE | DROP DATASHARE |
 CREATE LIBRARY | DROP LIBRARY |
 CREATE ROLE | DROP ROLE
 TRUNCATE TABLE
 VACUUM | ANALYZE | CANCEL }[, ...]
}
| { ALL [PRIVILEGES] }
FROM { ROLE role_name } [, ...]
```

### [Revoking explain permissions for row-level security policy filters](#)

The following is the syntax for revoking permissions to explain the row-level security policy filters of a query in the EXPLAIN plan. You can revoke the privilege using the REVOKE statement.

```
REVOKE EXPLAIN RLS FROM ROLE rolename
```

The following is the syntax for granting permissions to bypass row-level security policies for a query.

```
REVOKE IGNORE RLS FROM ROLE rolename
```

The following is the syntax for revoking permissions from the specified row-level security policy.

```
REVOKE SELECT ON [TABLE] table_name [, ...]
 FROM RLS POLICY policy_name [, ...]
```

## Parameters

### GRANT OPTION FOR

Revokes only the option to grant a specified permission to other users and doesn't revoke the permission itself. You can't revoke GRANT OPTION from a group or from PUBLIC.

### SELECT

Revokes the permission to select data from a table or a view using a SELECT statement.

### INSERT

Revokes the permission to load data into a table using an INSERT statement or a COPY statement.

### UPDATE

Revokes the permission to update a table column using an UPDATE statement.

**DELETE**

Revokes the permission to delete a data row from a table.

**REFERENCES**

Revokes the permission to create a foreign key constraint. You should revoke this permission on both the referenced table and the referencing table.

**ALL [ PRIVILEGES ]**

Revokes all available permissions at once from the specified user or group. The PRIVILEGES keyword is optional.

**ALTER**

Revokes permission to alter a table in an AWS Glue Data Catalog that is enabled for Lake Formation.  
This permission only applies when using Lake Formation.

**DROP**

Revokes permission to drop a table. This permission applies in Amazon Redshift and in an AWS Glue Data Catalog that is enabled for Lake Formation.

**ASSUMEROLE**

Revokes the permission to run COPY, UNLOAD, EXTERNAL FUNCTION, or CREATE MODEL commands from users, roles, or groups with a specified role.

**ON [ TABLE ] *table\_name***

Revokes the specified permissions on a table or a view. The TABLE keyword is optional.

**ON ALL TABLES IN SCHEMA *schema\_name***

Revokes the specified permissions on all tables in the referenced schema.

**( *column\_name* [...] ) ON TABLE *table\_name***

Revokes the specified permissions from users, groups, or PUBLIC on the specified columns of the Amazon Redshift table or view.

**( *column\_list* ) ON EXTERNAL TABLE *schema\_name.table\_name***

Revokes the specified permissions from an IAM role on the specified columns of the Lake Formation table in the referenced schema.

**ON EXTERNAL TABLE *schema\_name.table\_name***

Revokes the specified permissions from an IAM role on the specified Lake Formation tables in the referenced schema.

**ON EXTERNAL SCHEMA *schema\_name***

Revokes the specified permissions from an IAM role on the referenced schema.

**FROM IAM\_ROLE *iam\_role***

Indicates the IAM role losing the permissions.

**ROLE *role\_name***

Revokes the permissions from the specified role.

**GROUP *group\_name***

Revokes the permissions from the specified user group.

**PUBLIC**

Revokes the specified permissions from all users. PUBLIC represents a group that always includes all users. An individual user's permissions consist of the sum of permissions granted to PUBLIC,

permissions granted to any groups that the user belongs to, and any permissions granted to the user individually.

Revoking PUBLIC from a Lake Formation external table results in revoking the permission from the Lake Formation *everyone* group.

#### CREATE

Depending on the database object, revokes the following permissions from the user or group:

- For databases, using the CREATE clause for REVOKE prevents users from creating schemas within the database.
- For schemas, using the CREATE clause for REVOKE prevents users from creating objects within a schema. To rename an object, the user must have the CREATE permission and own the object to be renamed.

##### **Note**

By default, all users have CREATE and USAGE permissions on the PUBLIC schema.

#### TEMPORARY | TEMP

Revokes the permission to create temporary tables in the specified database.

##### **Note**

By default, users are granted permission to create temporary tables by their automatic membership in the PUBLIC group. To remove the permission for any users to create temporary tables, revoke the TEMP permission from the PUBLIC group and then explicitly grant the permission to create temporary tables to specific users or groups of users.

#### ON DATABASE *db\_name*

Revokes the permissions on the specified database.

#### USAGE

Revokes USAGE permissions on objects within a specific schema, which makes these objects inaccessible to users. Specific actions on these objects must be revoked separately (such as the EXECUTE permission on functions).

##### **Note**

By default, all users have CREATE and USAGE permissions on the PUBLIC schema.

#### ON SCHEMA *schema\_name*

Revokes the permissions on the specified schema. You can use schema permissions to control the creation of tables; the CREATE permission for a database only controls the creation of schemas.

#### RESTRICT

Revokes only those permissions that the user directly granted. This behavior is the default.

#### EXECUTE ON PROCEDURE *procedure\_name*

Revokes the EXECUTE permission on a specific stored procedure. Because stored procedure names can be overloaded, you must include the argument list for the procedure. For more information, see [Naming stored procedures \(p. 166\)](#).

#### EXECUTE ON ALL PROCEDURES IN SCHEMA *procedure\_name*

Revokes the specified permissions on all procedures in the referenced schema.

#### USAGE ON LANGUAGE *language\_name*

Revokes the USAGE permission on a language. For Python user-defined functions (UDFs), use plpythonu. For SQL UDFs, use sql. For stored procedures, use plpgsql.

To create a UDF, you must have permission for usage on language for SQL or plpythonu (Python). By default, USAGE ON LANGUAGE SQL is granted to PUBLIC. However, you must explicitly grant USAGE ON LANGUAGE PLPYTHONU to specific users or groups.

To revoke usage for SQL, first revoke usage from PUBLIC. Then grant usage on SQL only to the specific users or groups permitted to create SQL UDFs. The following example revokes usage on SQL from PUBLIC then grants usage to the user group udf\_devs.

```
revoke usage on language sql from PUBLIC;
grant usage on language sql to group udf_devs;
```

For more information, see [UDF security and privileges \(p. 150\)](#).

To revoke usage for stored procedures, first revoke usage from PUBLIC. Then grant usage on plpgsql only to the specific users or groups permitted to create stored procedures. For more information, see [Security and privileges for stored procedures \(p. 167\)](#).

**FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL } [, ...]**

Specifies the SQL command for which the permission is revoked. You can specify ALL to revoke the permission on the COPY, UNLOAD, EXTERNAL FUNCTION, and CREATE MODEL statements. This clause applies only to revoking the ASSUMEROLE permission.

**ALTER**

Revokes the ALTER permission for users or user groups that allows those that don't own a datashare to alter the datashare. This permission is required to add or remove objects from a datashare, or to set the property PUBLICACCESSIBLE. For more information, see [ALTER DATASHARE \(p. 629\)](#).

**SHARE**

Revokes permissions for users and user groups to add consumers to a datashare. Revoking this permission is required to stop the particular consumer from accessing the datashare from its clusters.

**ON DATASHARE *datashare\_name***

Grants the specified permissions on the referenced datashare.

**FROM *username***

Indicates the user losing the permissions.

**FROM GROUP *group\_name***

Indicates the user group losing the permissions.

**WITH GRANT OPTION**

Indicates that the user losing the permissions can in turn revoke the same permissions for others. You can't revoke WITH GRANT OPTION for a group or for PUBLIC.

**USAGE**

When USAGE is revoked for a consumer account or namespace within the same account, the specified consumer account or namespace within an account can't access the datashare and the objects of the datashare in read-only fashion.

Revoking the USAGE permission revokes the access to a datashare from consumers.

**FROM NAMESPACE 'clusternamespace GUID'**

Indicates the namespace in the same account that has consumers losing the permissions to the datashare. Namespaces use a 128-bit alphanumeric globally unique identifier (GUID).

**FROM ACCOUNT 'accountnumber' [ VIA DATA CATALOG ]**

Indicates the account number of another account that has the consumers losing the permissions to the datashare. Specifying 'VIA DATA CATALOG' indicates that you are revoking usage of the datashare from a Lake Formation account. Omitting the account number means that you're revoking from the account that owns the cluster.

ON DATABASE *shared\_database\_name* [, ...]

Revokes the specified usage permissions on the specified database that was created in the specified datashare.

ON SCHEMA *shared\_schema*

Revokes the specified permissions on the specified schema that was created in the specified datashare.

CREATE MODEL

Revokes the CREATE MODEL permission to create machine learning models in the specified database.

ON MODEL *model\_name*

Revokes the EXECUTE permission for a specific model.

[ADMIN OPTION FOR] { role } [, ...]

The role that you revoke from a specified user that has the WITH ADMIN OPTION.

FROM { role } [, ...]

The role that you revoke the specified role from.

## Usage notes

To learn more about the usage notes for REVOKE, see [the section called "Usage notes" \(p. 923\)](#).

## Examples

For examples of how to use REVOKE, see [the section called "Examples" \(p. 925\)](#).

## Usage notes

To revoke privileges from an object, you must meet one of the following criteria:

- Be the object owner.
- Be a superuser.
- Have a grant privilege for that object and privilege.

For example, the following command enables the user HR both to perform SELECT commands on the employees table and to grant and revoke the same privilege for other users.

```
grant select on table employees to HR with grant option;
```

HR can't revoke privileges for any operation other than SELECT, or on any other table than employees.

Superusers can access all objects regardless of GRANT and REVOKE commands that set object privileges.

PUBLIC represents a group that always includes all users. By default all members of PUBLIC have CREATE and USAGE privileges on the PUBLIC schema. To restrict any user's permissions on the PUBLIC schema, you must first revoke all permissions from PUBLIC on the PUBLIC schema, then grant privileges to specific users or groups. The following example controls table creation privileges in the PUBLIC schema.

```
revoke create on schema public from public;
```

To revoke privileges from a Lake Formation table, the IAM role associated with the table's external schema must have permission to revoke privileges to the external table. The following example creates an external schema with an associated IAM role `myGrantor`. IAM role `myGrantor` has the permission to revoke permissions from others. The REVOKE command uses the permission of the IAM role `myGrantor` that is associated with the external schema to revoke permission to the IAM role `myGrantee`.

```
create external schema mySchema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myGrantor'
create external database if not exists;
```

```
revoke select
on external table mySchema.mytable
from iam_role 'arn:aws:iam::123456789012:role/myGrantee';
```

#### **Note**

If the IAM role also has the ALL permission in an AWS Glue Data Catalog that is enabled for Lake Formation, the ALL permission isn't revoked. Only the SELECT permission is revoked. You can view the Lake Formation permissions in the Lake Formation console.

## Usage notes for revoking the ASSUMEROLE permission

The following usage notes apply to revoking the ASSUMEROLE privilege in Amazon Redshift.

Only a database superuser can revoke the ASSUMEROLE privilege for users and groups. A superuser always retains the ASSUMEROLE privilege.

To enable the use of the ASSUMEROLE privilege for users and groups, a superuser runs the following statement once on the cluster. Before granting the ASSUMEROLE privilege to users and groups, a superuser must run the following statement once on the cluster.

```
revoke assumerole on all from public for all;
```

## Usage notes for revoking machine learning permissions

You can't directly grant or revoke permissions related to an ML function. An ML function belongs to an ML model and permissions are controlled through the model. Instead, you can revoke permissions related to the ML model. The following example demonstrates how to revoke the run permission from all users associated with the model `customer_churn`.

```
REVOKE EXECUTE ON MODEL customer_churn FROM PUBLIC;
```

You can also revoke all permissions from a user for the ML model `customer_churn`.

```
REVOKE ALL on MODEL customer_churn FROM ml_user;
```

Granting or revoking the EXECUTE permission related to an ML function will fail if there is an ML function in the schema, even if that ML function already has the EXECUTE permission through GRANT EXECUTE ON MODEL. We recommend using a separate schema when using the CREATE MODEL command to keep the ML functions in a separate schema by themselves. The following example demonstrates how to do so.

```
CREATE MODEL ml_schema.customer_churn
```

```
FROM customer_data
TARGET churn
FUNCTION ml_schema.customer_churn_prediction
IAM_ROLE default
SETTINGS (
 S3_BUCKET 'your-s3-bucket'
);
```

## Examples

The following example revokes INSERT privileges on the SALES table from the GUESTS user group. This command prevents members of GUESTS from being able to load data into the SALES table by using the INSERT command.

```
revoke insert on table sales from group guests;
```

The following example revokes the SELECT privilege on all tables in the QA\_TICKIT schema from the user fred.

```
revoke select on all tables in schema qa_ticket from fred;
```

The following example revokes the privilege to select from a view for user bobr.

```
revoke select on table eventview from bobr;
```

The following example revokes the privilege to create temporary tables in the TICKIT database from all users.

```
revoke temporary on database ticket from public;
```

The following example revokes SELECT privilege on the cust\_name and cust\_phone columns of the cust\_profile table from the user user1.

```
revoke select(cust_name, cust_phone) on cust_profile from user1;
```

The following example revokes SELECT privilege on the cust\_name and cust\_phone columns and UPDATE privilege on the cust\_contact\_preference column of the cust\_profile table from the sales\_group group.

```
revoke select(cust_name, cust_phone), update(cust_contact_preference) on cust_profile from
group sales_group;
```

The following example shows the usage of the ALL keyword to revoke both SELECT and UPDATE privileges on three columns of the table cust\_profile from the sales\_admin group.

```
revoke ALL(cust_name, cust_phone,cust_contact_preference) on cust_profile from group
sales_admin;
```

The following example revokes the SELECT privilege on the cust\_name column of the cust\_profile\_vw view from the user2 user.

```
revoke select(cust_name) on cust_profile_vw from user2;
```

## Examples of revoking the USAGE privilege for datashares

The following example revokes access to the salesshare datashare for the Marketing namespace.

```
REVOKE USAGE ON DATASHARE salesshare FROM NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

The following example revokes the USAGE privilege on the sales\_db to Bob.

```
REVOKE USAGE ON DATABASE sales_db FROM Bob;
```

The following example REVOKE USAGE privilege on the sales\_schema to the Analyst\_group.

```
REVOKE USAGE ON SCHEMA sales_schema FROM GROUP Analyst_group;
```

## Examples of revoking the ASSUMEROLE privilege

The following are examples of revoking the ASSUMEROLE privilege.

A superuser must enable the use of the ASSUMEROLE privilege for users and groups by running the following statement once on the cluster:

```
revoke assumerole on all from public for all;
```

The following statement revokes the ASSUMEROLE privilege from user reg\_user1 on all roles for all operations.

```
revoke assumerole on all from reg_user1 for all;
```

## Examples of revoking the ROLE privilege

The following example revokes the sample\_role1 from to sample\_role2.

```
CREATE ROLE sample_role2;
GRANT ROLE sample_role1 TO ROLE sample_role2;
REVOKE ROLE sample_role1 FROM ROLE sample_role2;
```

The following example revokes system privileges from user1.

```
GRANT ROLE sys:DBA TO user1;
REVOKE ROLE sys:DBA FROM user1;
```

The following example revokes sample\_role1 and sample\_role2 from user1.

```
CREATE ROLE sample_role1;
CREATE ROLE sample_role2;
GRANT ROLE sample_role1, ROLE sample_role2 TO user1;
REVOKE ROLE sample_role1, ROLE sample_role2 FROM user1;
```

The following example revokes sample\_role2 with the ADMIN OPTION from user1.

```
GRANT ROLE sample_role2 TO user1 WITH ADMIN OPTION;
REVOKE ADMIN OPTION FOR ROLE sample_role2 FROM user1;
REVOKE ROLE sample_role2 FROM user1;
```

The following example revokes sample\_role1 and sample\_role2 from sample\_role5.

```
CREATE ROLE sample_role5;
GRANT ROLE sample_role1, ROLE sample_role2 TO ROLE sample_role5;
REVOKE ROLE sample_role1, ROLE sample_role2 FROM ROLE sample_role5;
```

The following example revokes the CREATE SCHEMA and DROP SCHEMA system privileges to sample\_role1.

```
GRANT CREATE SCHEMA, DROP SCHEMA TO ROLE sample_role1;
REVOKE CREATE SCHEMA, DROP SCHEMA FROM ROLE sample_role1;
```

## ROLLBACK

Stops the current transaction and discards all updates made by that transaction.

This command performs the same function as the [ABORT \(p. 626\)](#) command.

### Syntax

```
ROLLBACK [WORK | TRANSACTION]
```

### Parameters

WORK

Optional keyword. This keyword isn't supported within a stored procedure.

TRANSACTION

Optional keyword. WORK and TRANSACTION are synonyms. Neither is supported within a stored procedure.

For information about using ROLLBACK within a stored procedure, see [Managing transactions \(p. 169\)](#).

### Example

The following example creates a table then starts a transaction where data is inserted into the table. The ROLLBACK command then rolls back the data insertion to leave the table empty.

The following command creates an example table called MOVIE\_GROSS:

```
create table movie_gross(name varchar(30), gross bigint);
```

The next set of commands starts a transaction that inserts two data rows into the table:

```
begin;

insert into movie_gross values ('Raiders of the Lost Ark', 23400000);
insert into movie_gross values ('Star Wars', 10000000);
```

Next, the following command selects the data from the table to show that it was successfully inserted:

```
select * from movie_gross;
```

The command output shows that both rows successfully inserted:

| name                    | gross    |
|-------------------------|----------|
| Raiders of the Lost Ark | 23400000 |
| Star Wars               | 10000000 |
| (2 rows)                |          |

This command now rolls back the data changes to where the transaction began:

```
rollback;
```

Selecting data from the table now shows an empty table:

| name     | gross |
|----------|-------|
| (0 rows) |       |

## SELECT

### Topics

- [Syntax \(p. 928\)](#)
- [WITH clause \(p. 929\)](#)
- [SELECT list \(p. 934\)](#)
- [FROM clause \(p. 937\)](#)
- [WHERE clause \(p. 949\)](#)
- [GROUP BY clause \(p. 954\)](#)
- [HAVING clause \(p. 960\)](#)
- [UNION, INTERSECT, and EXCEPT \(p. 961\)](#)
- [ORDER BY clause \(p. 969\)](#)
- [CONNECT BY clause \(p. 971\)](#)
- [Subquery examples \(p. 973\)](#)
- [Correlated subqueries \(p. 974\)](#)

Returns rows from tables, views, and user-defined functions.

### Note

The maximum size for a single SQL statement is 16 MB.

## Syntax

```
[WITH with_subquery [, ...]]
SELECT
[TOP number | [ALL | DISTINCT]
* | expression [AS output_name] [, ...]]
[FROM table_reference [, ...]]
[WHERE condition]
[[START WITH expression] CONNECT BY expression]
```

```
[GROUP BY expression [, ...]]
[HAVING condition]
[{ UNION | ALL | INTERSECT | EXCEPT | MINUS } query]
[ORDER BY expression [ASC | DESC]]
[LIMIT { number | ALL }]
[OFFSET start]
```

## WITH clause

A WITH clause is an optional clause that precedes the SELECT list in a query. The WITH clause defines one or more *common\_table\_expressions*. Each common table expression (CTE) defines a temporary table, which is similar to a view definition. You can reference these temporary tables in the FROM clause. They're used only while the query they belong to runs. Each CTE in the WITH clause specifies a table name, an optional list of column names, and a query expression that evaluates to a table (a SELECT statement). When you reference the temporary table name in the FROM clause of the same query expression that defines it, the CTE is recursive.

WITH clause subqueries are an efficient way of defining tables that can be used throughout the execution of a single query. In all cases, the same results can be achieved by using subqueries in the main body of the SELECT statement, but WITH clause subqueries may be simpler to write and read. Where possible, WITH clause subqueries that are referenced multiple times are optimized as common subexpressions; that is, it may be possible to evaluate a WITH subquery once and reuse its results. (Note that common subexpressions aren't limited to those defined in the WITH clause.)

### Syntax

```
[WITH [RECURSIVE] common_table_expression [, common_table_expression , ...]]
```

where *common\_table\_expression* can be either non-recursive or recursive. Following is the non-recursive form:

```
CTE_table_name [(column_name [, ...])] AS (query)
```

Following is the recursive form of *common\_table\_expression*:

```
CTE_table_name (column_name [, ...]) AS (recursive_query)
```

### Parameters

#### RECURSIVE

Keyword that identifies the query as a recursive CTE. This keyword is required if any *common\_table\_expression* defined in the WITH clause is recursive. You can only specify the RECURSIVE keyword once, immediately following the WITH keyword, even when the WITH clause contains multiple recursive CTEs. In general, a recursive CTE is a UNION ALL subquery with two parts.

#### *common\_table\_expression*

Defines a temporary table that you can reference in the [FROM clause \(p. 937\)](#) and is used only during the execution of the query to which it belongs.

#### *CTE\_table\_name*

A unique name for a temporary table that defines the results of a WITH clause subquery. You can't use duplicate names within a single WITH clause. Each subquery must be given a table name that can be referenced in the [FROM clause \(p. 937\)](#).

*column\_name*

A list of output column names for the WITH clause subquery, separated by commas. The number of column names specified must be equal to or less than the number of columns defined by the subquery. For a CTE that is non-recursive, the *column\_name* clause is optional. For a recursive CTE, the *column\_name* list is required.

*query*

Any SELECT query that Amazon Redshift supports. See [SELECT \(p. 928\)](#).

*recursive\_query*

A UNION ALL query that consists of two SELECT subqueries:

- The first SELECT subquery doesn't have a recursive reference to the same *CTE\_table\_name*. It returns a result set that is the initial seed of the recursion. This part is called the initial member or seed member.
- The second SELECT subquery references the same *CTE\_table\_name* in its FROM clause. This is called the recursive member. The *recursive\_query* contains a WHERE condition to end the *recursive\_query*.

## Usage notes

You can use a WITH clause in the following SQL statements:

- SELECT
- SELECT INTO
- CREATE TABLE AS
- CREATE VIEW
- DECLARE
- EXPLAIN
- INSERT INTO...SELECT
- PREPARE
- UPDATE (within a WHERE clause subquery. You can't define a recursive CTE in the subquery. The recursive CTE must precede the UPDATE clause.)
- DELETE

If the FROM clause of a query that contains a WITH clause doesn't reference any of the tables defined by the WITH clause, the WITH clause is ignored and the query runs as normal.

A table defined by a WITH clause subquery can be referenced only in the scope of the SELECT query that the WITH clause begins. For example, you can reference such a table in the FROM clause of a subquery in the SELECT list, WHERE clause, or HAVING clause. You can't use a WITH clause in a subquery and reference its table in the FROM clause of the main query or another subquery. This query pattern results in an error message of the form `relation table_name doesn't exist` for the WITH clause table.

You can't specify another WITH clause inside a WITH clause subquery.

You can't make forward references to tables defined by WITH clause subqueries. For example, the following query returns an error because of the forward reference to table W2 in the definition of table W1:

```
with w1 as (select * from w2), w2 as (select * from w1)
select * from sales;
ERROR: relation "w2" does not exist
```

A WITH clause subquery may not consist of a SELECT INTO statement; however, you can use a WITH clause in a SELECT INTO statement.

## Recursive common table expressions

A recursive *common table expression (CTE)* is a CTE that references itself. A recursive CTE is useful in querying hierarchical data, such as organization charts that show reporting relationships between employees and managers. See [Example: Recursive CTE \(p. 933\)](#).

Another common use is a multilevel bill of materials, when a product consists of many components and each component itself also consists of other components or subassemblies.

Be sure to limit the depth of recursion by including a WHERE clause in the second SELECT subquery of the recursive query. For an example, see [Example: Recursive CTE \(p. 933\)](#). Otherwise, an error can occur similar to the following:

- Recursive CTE out of working buffers.
- Exceeded recursive CTE max\_rows limit, please add correct CTE termination predicates or change the max\_recursion\_rows parameter.

### Note

`max_recursion_rows` is a parameter setting the maximum number of rows a recursive CTE can return in order to prevent infinite recursion loops. We recommend against changing this to a larger value than the default. This prevents infinite recursion problems in your queries from taking up excessive space in your cluster.

You can specify a sort order and limit on the result of the recursive CTE. You can include group by and distinct options on the final result of the recursive CTE.

You can't specify a WITH RECURSIVE clause inside a subquery. The `recursive_query` member can't include an order by or limit clause.

## Examples

The following example shows the simplest possible case of a query that contains a WITH clause. The WITH query named VENUECOPY selects all of the rows from the VENUE table. The main query in turn selects all of the rows from VENUECOPY. The VENUECOPY table exists only for the duration of this query.

```
with venuecopy as (select * from venue)
select * from venuecopy order by 1 limit 10;
```

| venueid   | venuename                  | venuecity       | venuestate | venueseats |
|-----------|----------------------------|-----------------|------------|------------|
| 1         | Toyota Park                | Bridgeview      | IL         | 0          |
| 2         | Columbus Crew Stadium      | Columbus        | OH         | 0          |
| 3         | RFK Stadium                | Washington      | DC         | 0          |
| 4         | CommunityAmerica Ballpark  | Kansas City     | KS         | 0          |
| 5         | Gillette Stadium           | Foxborough      | MA         | 68756      |
| 6         | New York Giants Stadium    | East Rutherford | NJ         | 80242      |
| 7         | BMO Field                  | Toronto         | ON         | 0          |
| 8         | The Home Depot Center      | Carson          | CA         | 0          |
| 9         | Dick's Sporting Goods Park | Commerce City   | CO         | 0          |
| v 10      | Pizza Hut Park             | Frisco          | TX         | 0          |
| (10 rows) |                            |                 |            |            |

The following example shows a WITH clause that produces two tables, named VENUE\_SALES and TOP\_VENUES. The second WITH query table selects from the first. In turn, the WHERE clause of the main query block contains a subquery that constrains the TOP\_VENUES table.

```

with venue_sales as
(select venuename, venuecity, sum(pricepaid) as venuename_sales
from sales, venue, event
where venue.venueid=event.venueid and event.eventid=sales.eventid
group by venuename, venuecity),

top_venues as
(select venuename
from venue_sales
where venuename_sales > 800000)

select venuename, venuecity, venuestate,
sum(qtysold) as venue_qty,
sum(pricepaid) as venue_sales
from sales, venue, event
where venue.venueid=event.venueid and event.eventid=sales.eventid
and venuename in(select venuename from top_venues)
group by venuename, venuecity, venuestate
order by venuename;

```

| venuename               | venuecity     | venuestate | venue_qty | venue_sales |
|-------------------------|---------------|------------|-----------|-------------|
| August Wilson Theatre   | New York City | NY         | 3187      | 1032156.00  |
| Biltmore Theatre        | New York City | NY         | 2629      | 828981.00   |
| Charles Playhouse       | Boston        | MA         | 2502      | 857031.00   |
| Ethel Barrymore Theatre | New York City | NY         | 2828      | 891172.00   |
| Eugene O'Neill Theatre  | New York City | NY         | 2488      | 828950.00   |
| Greek Theatre           | Los Angeles   | CA         | 2445      | 838918.00   |
| Helen Hayes Theatre     | New York City | NY         | 2948      | 978765.00   |
| Hilton Theatre          | New York City | NY         | 2999      | 885686.00   |
| Imperial Theatre        | New York City | NY         | 2702      | 877993.00   |
| Lunt-Fontanne Theatre   | New York City | NY         | 3326      | 1115182.00  |
| Majestic Theatre        | New York City | NY         | 2549      | 894275.00   |
| Nederlander Theatre     | New York City | NY         | 2934      | 936312.00   |
| Pasadena Playhouse      | Pasadena      | CA         | 2739      | 820435.00   |
| Winter Garden Theatre   | New York City | NY         | 2838      | 939257.00   |
| (14 rows)               |               |            |           |             |

The following two examples demonstrate the rules for the scope of table references based on WITH clause subqueries. The first query runs, but the second fails with an expected error. The first query has WITH clause subquery inside the SELECT list of the main query. The table defined by the WITH clause (HOLIDAYS) is referenced in the FROM clause of the subquery in the SELECT list:

```

select caldate, sum(pricepaid) as daysales,
(with holidays as (select * from date where holiday = 't')
select sum(pricepaid)
from sales join holidays on sales.dateid=holidays.dateid
where caldate='2008-12-25') as dec25sales
from sales join date on sales.dateid=date.dateid
where caldate in('2008-12-25','2008-12-31')
group by caldate
order by caldate;

caldate | daysales | dec25sales
-----+-----+-----
2008-12-25 | 70402.00 | 70402.00
2008-12-31 | 12678.00 | 70402.00
(2 rows)

```

The second query fails because it attempts to reference the HOLIDAYS table in the main query as well as in the SELECT list subquery. The main query references are out of scope.

```

select caldate, sum(pricepaid) as daysales,
(with holidays as (select * from date where holiday ='t')
select sum(pricepaid)
from sales join holidays on sales.dateid=holidays.dateid
where caldate='2008-12-25') as dec25sales
from sales join holidays on sales.dateid=holidays.dateid
where caldate in('2008-12-25','2008-12-31')
group by caldate
order by caldate;

ERROR: relation "holidays" does not exist

```

## Example: Recursive CTE

The following is an example of a recursive CTE that returns the employees who report directly or indirectly to John. The recursive query contains a WHERE clause to limit the depth of recursion to less than 4 levels.

```

--create and populate the sample table
create table employee (
 id int,
 name varchar (20),
 manager_id int
);

insert into employee(id, name, manager_id) values
(100, 'Carlos', null),
(101, 'John', 100),
(102, 'Jorge', 101),
(103, 'Kwaku', 101),
(110, 'Liu', 101),
(106, 'Mateo', 102),
(110, 'Nikki', 103),
(104, 'Paulo', 103),
(105, 'Richard', 103),
(120, 'Saanvi', 104),
(200, 'Shirley', 104),
(201, 'Sofia', 102),
(205, 'Zhang', 104);

--run the recursive query
with recursive john_org(id, name, manager_id, level) as
(select id, name, manager_id, 1 as level
 from employee
 where name = 'John'
 union all
 select e.id, e.name, e.manager_id, level + 1 as next_level
 from employee e, john_org j
 where e.manager_id = j.id and level < 4
)
 select distinct id, name, manager_id from john_org order by manager_id;

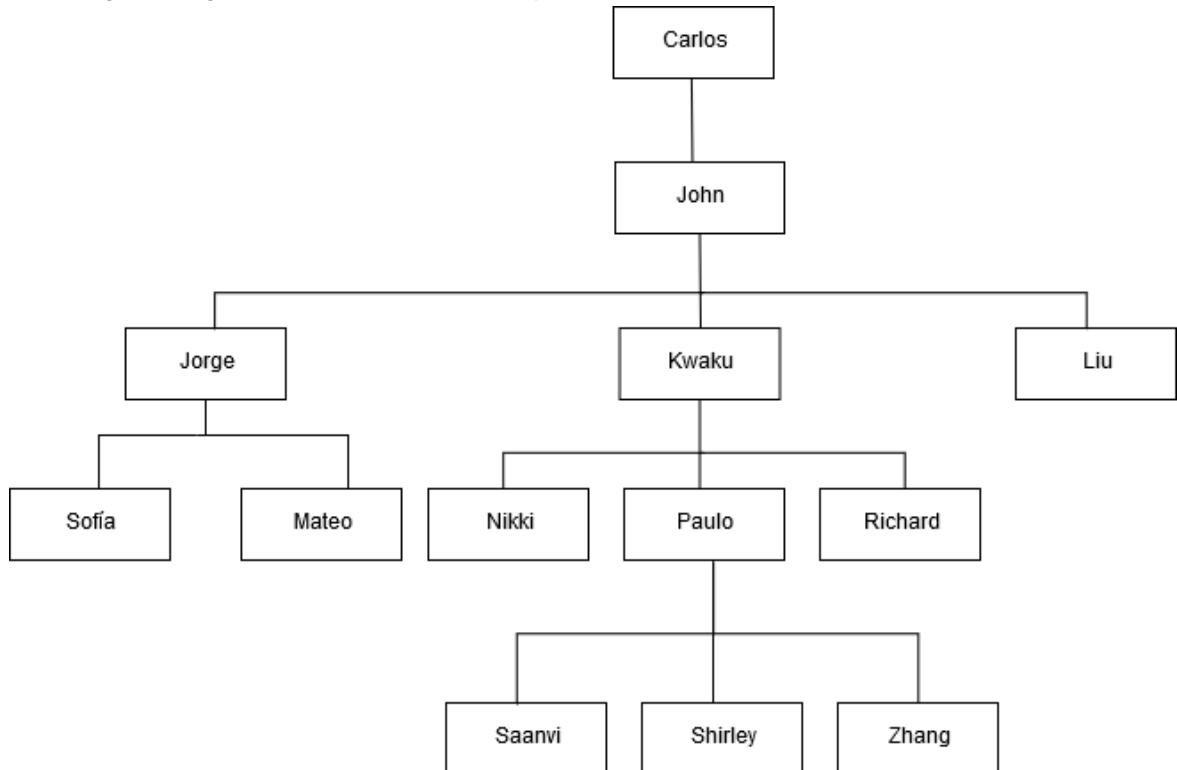
```

Following is the result of the query.

| id  | name  | manager_id |
|-----|-------|------------|
| 101 | John  | 100        |
| 102 | Jorge | 101        |
| 103 | Kwaku | 101        |
| 110 | Liu   | 101        |
| 201 | Sofia | 102        |
| 106 | Mateo | 102        |

|     |         |     |
|-----|---------|-----|
| 110 | Nikki   | 103 |
| 104 | Paulo   | 103 |
| 105 | Richard | 103 |
| 120 | Saanvi  | 104 |
| 200 | Shirley | 104 |
| 205 | Zhang   | 104 |

Following is an organization chart for John's department.



## SELECT list

### Topics

- [Syntax \(p. 934\)](#)
- [Parameters \(p. 935\)](#)
- [Usage notes \(p. 935\)](#)
- [Examples \(p. 936\)](#)

The SELECT list names the columns, functions, and expressions that you want the query to return. The list represents the output of the query.

For more information about SQL functions, see [SQL functions reference \(p. 1020\)](#). For more information about expressions, see [Conditional expressions \(p. 1056\)](#).

### Syntax

```

SELECT
[TOP number]
[ALL | DISTINCT] * | expression [AS column_alias] [, ...]

```

## Parameters

### TOP *number*

TOP takes a positive integer as its argument, which defines the number of rows that are returned to the client. The behavior with the TOP clause is the same as the behavior with the LIMIT clause. The number of rows that is returned is fixed, but the set of rows isn't. To return a consistent set of rows, use TOP or LIMIT in conjunction with an ORDER BY clause.

### ALL

A redundant keyword that defines the default behavior if you don't specify DISTINCT. SELECT ALL \* means the same as SELECT \* (select all rows for all columns and retain duplicates).

### DISTINCT

Option that eliminates duplicate rows from the result set, based on matching values in one or more columns.

### \* (asterisk)

Returns the entire contents of the table (all columns and all rows).

### *expression*

An expression formed from one or more columns that exist in the tables referenced by the query. An expression can contain SQL functions. For example:

```
avg(datediff(day, listtime, saletime))
```

### AS *column\_alias*

A temporary name for the column that is used in the final result set. The AS keyword is optional. For example:

```
avg(datediff(day, listtime, saletime)) as avgwait
```

If you don't specify an alias for an expression that isn't a simple column name, the result set applies a default name to that column.

#### Note

The alias is recognized right after it is defined in the target list. You can use an alias in other expressions defined after it in the same target list. The following example illustrates this.

```
select clicks / impressions as probability, round(100 * probability, 1) as percentage from raw_data;
```

The benefit of the lateral alias reference is you don't need to repeat the aliased expression when building more complex expressions in the same target list. When Amazon Redshift parses this type of reference, it just inlines the previously defined aliases. If there is a column with the same name defined in the FROM clause as the previously aliased expression, the column in the FROM clause takes priority. For example, in the above query if there is a column named 'probability' in table raw\_data, the 'probability' in the second expression in the target list refers to that column instead of the alias name 'probability'.

## Usage notes

TOP is a SQL extension; it provides an alternative to the LIMIT behavior. You can't use TOP and LIMIT in the same query.

## Examples

The following example returns 10 rows from the SALES table. Though the query uses the TOP clause, it still returns an unpredictable set of rows because no ORDER BY clause is specified,

```
select top 10 *
from sales;
```

The following query is functionally equivalent, but uses a LIMIT clause instead of a TOP clause:

```
select *
from sales
limit 10;
```

The following example returns the first 10 rows from the SALES table using the TOP clause, ordered by the QTYSOLD column in descending order.

```
select top 10 qtysold, sellerid
from sales
order by qtysold desc, sellerid;

qtysold | sellerid
-----+-----
8 | 518
8 | 520
8 | 574
8 | 718
8 | 868
8 | 2663
8 | 3396
8 | 3726
8 | 5250
8 | 6216
(10 rows)
```

The following example returns the first two QTYSOLD and SELLERID values from the SALES table, ordered by the QTYSOLD column:

```
select top 2 qtysold, sellerid
from sales
order by qtysold desc, sellerid;

qtysold | sellerid
-----+-----
8 | 518
8 | 520
(2 rows)
```

The following example shows the list of distinct category groups from the CATEGORY table:

```
select distinct catgroup from category
order by 1;

catgroup

Concerts
Shows
Sports
(3 rows)
```

```
--the same query, run without distinct
select catgroup from category
order by 1;

catgroup

Concerts
Concerts
Concerts
Shows
Shows
Shows
Sports
Sports
Sports
Sports
Sports
Sports
Sports
(11 rows)
```

The following example returns the distinct set of week numbers for December 2008. Without the DISTINCT clause, the statement would return 31 rows, or one for each day of the month.

```
select distinct week, month, year
from date
where month='DEC' and year=2008
order by 1, 2, 3;

week | month | year
-----+-----+-----
49 | DEC | 2008
50 | DEC | 2008
51 | DEC | 2008
52 | DEC | 2008
53 | DEC | 2008
(5 rows)
```

## FROM clause

The FROM clause in a query lists the table references (tables, views, and subqueries) that data is selected from. If multiple table references are listed, the tables must be joined, using appropriate syntax in either the FROM clause or the WHERE clause. If no join criteria are specified, the system processes the query as a cross-join (Cartesian product).

### Topics

- [Syntax \(p. 937\)](#)
- [Parameters \(p. 938\)](#)
- [Usage notes \(p. 939\)](#)
- [PIVOT and UNPIVOT examples \(p. 940\)](#)
- [JOIN examples \(p. 946\)](#)

### Syntax

```
FROM table_reference [, ...]
```

where *table\_reference* is one of the following:

```

with_subquery_table_name [[AS] alias [(column_alias [, ...])]]
table_name [*] [[AS] alias [(column_alias [, ...])]]
(subquery) [AS] alias [(column_alias [, ...])]
table_reference [NATURAL] join_type table_reference
[ON join_condition | USING (join_column [, ...])]
table_reference [PIVOT] (aggregate(expr) [[AS] aggregate_alias]
FOR column_name IN (expression [[AS] in_alias [, ...]
)
) [[AS] in_alias [, ...])]
table_reference [UNPIVOT [INCLUDE NULLS | EXCLUDE NULLS]] (value_column_name
FOR name_column_name
IN (column_reference [[AS] in_alias [, ...]
)
) [[AS] alias [(column_alias, ...))]

```

## Parameters

*with\_subquery\_table\_name*

A table defined by a subquery in the [WITH clause \(p. 929\)](#).

*table\_name*

Name of a table or view.

*alias*

Temporary alternative name for a table or view. An alias must be supplied for a table derived from a subquery. In other table references, aliases are optional. The AS keyword is always optional. Table aliases provide a convenient shortcut for identifying tables in other parts of a query, such as the WHERE clause. For example:

```
select * from sales s, listing l
where s.listid=l.listid
```

*column\_alias*

Temporary alternative name for a column in a table or view.

*subquery*

A query expression that evaluates to a table. The table exists only for the duration of the query and is typically given a name or *alias*. However, an alias isn't required. You can also define column names for tables that derive from subqueries. Naming column aliases is important when you want to join the results of subqueries to other tables and when you want to select or constrain those columns elsewhere in the query.

A subquery may contain an ORDER BY clause, but this clause may have no effect if a LIMIT or OFFSET clause isn't also specified.

**PIVOT**

Rotates output from rows to columns, for the purpose of representing tabular data in a format that is easy to read. Output is represented horizontally across multiple columns. PIVOT is similar to a GROUP BY query with an aggregation, using an aggregate expression to specify an output format. However, in contrast to GROUP BY, the results are returned in columns instead of rows.

For examples that show how to query with PIVOT and UNPIVOT, see [PIVOT and UNPIVOT examples \(p. 940\)](#).

**UNPIVOT**

The UNPIVOT operator transforms result columns, from an input table or query results, into rows, to make the output easier to read. UNPIVOT combines the data of its input columns into two result

columns: a name column and a value column. The name column contains column names from the input, as row entries. The value column contains values from the input columns, such as results of an aggregation. For example, the counts of items in various categories.

For examples that show how to query semistructured data, such as JSON-formatted data, using UNPIVOT, see [Object unpivoting \(p. 379\)](#).

#### NATURAL

Defines a join that automatically uses all pairs of identically named columns in the two tables as the joining columns. No explicit join condition is required. For example, if the CATEGORY and EVENT tables both have columns named CATID, a natural join of those tables is a join over their CATID columns.

##### Note

If a NATURAL join is specified but no identically named pairs of columns exist in the tables to be joined, the query defaults to a cross-join.

#### *join\_type*

Specify one of the following types of join:

- [INNER] JOIN
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN
- FULL [OUTER] JOIN
- CROSS JOIN

Cross-joins are unqualified joins; they return the Cartesian product of the two tables.

Inner and outer joins are qualified joins. They are qualified either implicitly (in natural joins); with the ON or USING syntax in the FROM clause; or with a WHERE clause condition.

An inner join returns matching rows only, based on the join condition or list of joining columns. An outer join returns all of the rows that the equivalent inner join would return plus non-matching rows from the "left" table, "right" table, or both tables. The left table is the first-listed table, and the right table is the second-listed table. The non-matching rows contain NULL values to fill the gaps in the output columns.

#### ON *join\_condition*

Type of join specification where the joining columns are stated as a condition that follows the ON keyword. For example:

```
sales join listing
on sales.listid=listlisting.listid and sales.eventid=listlisting.eventid
```

#### USING (*join\_column* [, ...])

Type of join specification where the joining columns are listed in parentheses. If multiple joining columns are specified, they are delimited by commas. The USING keyword must precede the list. For example:

```
sales join listing
using (listid,eventid)
```

## Usage notes

Joining columns must have comparable data types.

A NATURAL or USING join retains only one of each pair of joining columns in the intermediate result set.

A join with the ON syntax retains both joining columns in its intermediate result set.

See also [WITH clause \(p. 929\)](#).

## PIVOT and UNPIVOT examples

PIVOT and UNPIVOT are parameters in the FROM clause that rotate query output from rows to columns and columns to rows, respectively. They represent tabular query results in a format that's easy to read. The following examples use test data and queries to show how to use them.

For more information about these and other parameters, see [FROM clause](#).

### PIVOT examples

Set up the sample table and data and use them to run the subsequent example queries.

```
CREATE TABLE part (
 partname varchar,
 manufacturer varchar,
 quality int,
 price decimal(12, 2)
);

INSERT INTO part VALUES ('prop', 'local parts co', 2, 10.00);
INSERT INTO part VALUES ('prop', 'big parts co', NULL, 9.00);
INSERT INTO part VALUES ('prop', 'small parts co', 1, 12.00);

INSERT INTO part VALUES ('rudder', 'local parts co', 1, 2.50);
INSERT INTO part VALUES ('rudder', 'big parts co', 2, 3.75);
INSERT INTO part VALUES ('rudder', 'small parts co', NULL, 1.90);

INSERT INTO part VALUES ('wing', 'local parts co', NULL, 7.50);
INSERT INTO part VALUES ('wing', 'big parts co', 1, 15.20);
INSERT INTO part VALUES ('wing', 'small parts co', NULL, 11.80);
```

PIVOT on partname with an AVG aggregation on price.

```
SELECT *
FROM (SELECT partname, price FROM part) PIVOT (
 AVG(price) FOR partname IN ('prop', 'rudder', 'wing')
);
```

The query results in the following output.

| prop  | rudder | wing  |
|-------|--------|-------|
| 10.33 | 2.71   | 11.50 |

In the previous example, the results are transformed into columns. The following example shows a GROUP BY query that returns the average prices in rows, rather than in columns.

```
SELECT partname, avg(price)
FROM (SELECT partname, price FROM part)
WHERE partname IN ('prop', 'rudder', 'wing')
GROUP BY partname;
```

The query results in the following output.

| partname | avg |
|----------|-----|
|----------|-----|

|        |  |       |
|--------|--|-------|
| prop   |  | 10.33 |
| rudder |  | 2.71  |
| wing   |  | 11.50 |

A PIVOT example with manufacturer as an implicit column.

```
SELECT *
FROM (SELECT quality, manufacturer FROM part) PIVOT (
 count(*) FOR quality IN (1, 2, NULL)
);
```

The query results in the following output.

| manufacturer   |  | 1 |  | 2 |  | null |
|----------------|--|---|--|---|--|------|
| local parts co |  | 1 |  | 1 |  | 1    |
| big parts co   |  | 1 |  | 1 |  | 1    |
| small parts co |  | 1 |  | 0 |  | 2    |

Input table columns that are not referenced in the PIVOT definition are added implicitly to the result table. This is the case for the manufacturer column in the previous example. The example also shows that NULL is a valid value for the IN operator.

PIVOT in the above example returns similar information as the following query, which includes GROUP BY. The difference is that PIVOT returns the value 0 for column 2 and the manufacturer small parts co. The GROUP BY query does not contain a corresponding row. In most cases, PIVOT inserts NULL if a row doesn't have input data for a given column. However, the count aggregate doesn't return NULL and 0 is the default value.

```
SELECT manufacturer, quality, count(*)
FROM (SELECT quality, manufacturer FROM part)
WHERE quality IN (1, 2) OR quality IS NULL
GROUP BY manufacturer, quality
ORDER BY manufacturer;
```

The query results in the following output.

| manufacturer   |  | quality |  | count |
|----------------|--|---------|--|-------|
| big parts co   |  |         |  | 1     |
| big parts co   |  | 2       |  | 1     |
| big parts co   |  | 1       |  | 1     |
| local parts co |  | 2       |  | 1     |
| local parts co |  | 1       |  | 1     |
| local parts co |  |         |  | 1     |
| small parts co |  | 1       |  | 1     |
| small parts co |  |         |  | 2     |

The PIVOT operator accepts optional aliases on the aggregate expression and on each value for the IN operator. Use aliases to customize the column names. If there is no aggregate alias, only the IN list aliases are used. Otherwise, the aggregate alias is appended to the column name with an underscore to separate the names.

```
SELECT *
FROM (SELECT quality, manufacturer FROM part) PIVOT (
 count(*) AS count FOR quality IN (1 AS high, 2 AS low, NULL AS na)
);
```

The query results in the following output.

| manufacturer   | high_count | low_count | na_count |
|----------------|------------|-----------|----------|
| local parts co | 1          | 1         | 1        |
| big parts co   | 1          | 1         | 1        |
| small parts co | 1          | 0         | 2        |

Set up the following sample table and data and use them to run the subsequent example queries. The data represents booking dates for a collection of hotels.

```

CREATE TABLE bookings (
 booking_id int,
 hotel_code char(8),
 booking_date date,
 price decimal(12, 2)
);

INSERT INTO bookings VALUES (1, 'FOREST_L', '02/01/2023', 75.12);
INSERT INTO bookings VALUES (2, 'FOREST_L', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (3, 'FOREST_L', '02/04/2023', 85.54);

INSERT INTO bookings VALUES (4, 'FOREST_L', '02/08/2023', 75.00);
INSERT INTO bookings VALUES (5, 'FOREST_L', '02/11/2023', 75.00);
INSERT INTO bookings VALUES (6, 'FOREST_L', '02/14/2023', 90.00);

INSERT INTO bookings VALUES (7, 'FOREST_L', '02/21/2023', 60.00);
INSERT INTO bookings VALUES (8, 'FOREST_L', '02/22/2023', 85.00);
INSERT INTO bookings VALUES (9, 'FOREST_L', '02/27/2023', 90.00);

INSERT INTO bookings VALUES (10, 'DESERT_S', '02/01/2023', 98.00);
INSERT INTO bookings VALUES (11, 'DESERT_S', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (12, 'DESERT_S', '02/04/2023', 85.00);

INSERT INTO bookings VALUES (13, 'DESERT_S', '02/05/2023', 75.00);
INSERT INTO bookings VALUES (14, 'DESERT_S', '02/06/2023', 34.00);
INSERT INTO bookings VALUES (15, 'DESERT_S', '02/09/2023', 85.00);

INSERT INTO bookings VALUES (16, 'DESERT_S', '02/12/2023', 23.00);
INSERT INTO bookings VALUES (17, 'DESERT_S', '02/13/2023', 76.00);
INSERT INTO bookings VALUES (18, 'DESERT_S', '02/14/2023', 85.00);

INSERT INTO bookings VALUES (19, 'OCEAN_WV', '02/01/2023', 98.00);
INSERT INTO bookings VALUES (20, 'OCEAN_WV', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (21, 'OCEAN_WV', '02/04/2023', 85.00);

INSERT INTO bookings VALUES (22, 'OCEAN_WV', '02/06/2023', 75.00);
INSERT INTO bookings VALUES (23, 'OCEAN_WV', '02/09/2023', 34.00);
INSERT INTO bookings VALUES (24, 'OCEAN_WV', '02/12/2023', 85.00);

INSERT INTO bookings VALUES (25, 'OCEAN_WV', '02/13/2023', 23.00);
INSERT INTO bookings VALUES (26, 'OCEAN_WV', '02/14/2023', 76.00);
INSERT INTO bookings VALUES (27, 'OCEAN_WV', '02/16/2023', 85.00);

INSERT INTO bookings VALUES (28, 'CITY_BLD', '02/01/2023', 98.00);
INSERT INTO bookings VALUES (29, 'CITY_BLD', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (30, 'CITY_BLD', '02/04/2023', 85.00);

INSERT INTO bookings VALUES (31, 'CITY_BLD', '02/12/2023', 75.00);
INSERT INTO bookings VALUES (32, 'CITY_BLD', '02/13/2023', 34.00);
INSERT INTO bookings VALUES (33, 'CITY_BLD', '02/17/2023', 85.00);

INSERT INTO bookings VALUES (34, 'CITY_BLD', '02/22/2023', 23.00);
INSERT INTO bookings VALUES (35, 'CITY_BLD', '02/23/2023', 76.00);

```

```
INSERT INTO bookings VALUES (36, 'CITY_BLD', '02/24/2023', 85.00);
```

In this sample query, booking records are tallied to give a total for each week. The end date for each week becomes a column name.

```
SELECT * FROM
 (SELECT
 booking_id,
 (date_trunc('week', booking_date::date) + '5 days'::interval)::date as enddate,
 hotel_code AS "hotel code"
 FROM bookings
) PIVOT (
 count(booking_id) FOR enddate IN ('2023-02-04','2023-02-11','2023-02-18')
);
```

The query results in the following output.

| hotel code | 2023-02-04 | 2023-02-11 | 2023-02-18 |
|------------|------------|------------|------------|
| FOREST_L   | 3          | 2          | 1          |
| DESERT_S   | 4          | 3          | 2          |
| OCEAN_WV   | 3          | 3          | 3          |
| CITY_BLD   | 3          | 1          | 2          |

Amazon Redshift doesn't support CROSSTAB to pivot on multiple columns. But you can change row data to columns, in a similar manner to an aggregation with PIVOT, with a query like the following. This uses the same booking sample data as the previous example.

```
SELECT
 booking_date,
 MAX(CASE WHEN hotel_code = 'FOREST_L' THEN 'forest is booked' ELSE '' END) AS FOREST_L,
 MAX(CASE WHEN hotel_code = 'DESERT_S' THEN 'desert is booked' ELSE '' END) AS DESERT_S,
 MAX(CASE WHEN hotel_code = 'OCEAN_WV' THEN 'ocean is booked' ELSE '' END) AS OCEAN_WV
FROM bookings
GROUP BY booking_date
ORDER BY booking_date asc;
```

The sample query results in booking dates listed next to short phrases that indicate which hotels are booked.

| booking_date | forest_l         | desert_s         | ocean_wv        |
|--------------|------------------|------------------|-----------------|
| 2023-02-01   | forest is booked | desert is booked | ocean is booked |
| 2023-02-02   | forest is booked | desert is booked | ocean is booked |
| 2023-02-04   | forest is booked | desert is booked | ocean is booked |
| 2023-02-05   |                  | desert is booked |                 |
| 2023-02-06   |                  | desert is booked |                 |

The following are usage notes for PIVOT:

- PIVOT can be applied to tables, sub-queries, and common table expressions (CTEs). PIVOT cannot be applied to any JOIN expressions, recursive CTEs, PIVOT, or UNPIVOT expressions. Also not supported are SUPER unnested expressions and Redshift Spectrum nested tables.
- PIVOT supports the COUNT, SUM, MIN, MAX, and AVG aggregate functions.
- The PIVOT aggregate expression has to be a call of a supported aggregate function. Complex expressions on top of the aggregate are not supported. The aggregate arguments cannot contain references to tables other than the PIVOT input table. Correlated references to a parent query are also

not supported. The aggregate argument may contain sub-queries. These can be correlated internally or on the PIVOT input table.

- The PIVOT IN list values cannot be column references or sub-queries. Each value must be type compatible with the FOR column reference.
- If the IN list values do not have aliases, PIVOT generates default column names. For constant IN values such as 'abc' or 5 the default column name is the constant itself. For any complex expression, the column name is a standard Amazon Redshift default name such as ?column?.

### UNPIVOT examples

Set up the sample data and use it to run the subsequent examples.

```
CREATE TABLE count_by_color (quality varchar, red int, green int, blue int);

INSERT INTO count_by_color VALUES ('high', 15, 20, 7);
INSERT INTO count_by_color VALUES ('normal', 35, NULL, 40);
INSERT INTO count_by_color VALUES ('low', 10, 23, NULL);
```

UNPIVOT on input columns red, green, and blue.

```
SELECT *
FROM (SELECT red, green, blue FROM count_by_color) UNPIVOT (
 cnt FOR color IN (red, green, blue)
);
```

The query results in the following output.

| color | cnt |
|-------|-----|
| red   | 15  |
| red   | 35  |
| red   | 10  |
| green | 20  |
| green | 23  |
| blue  | 7   |
| blue  | 40  |

By default, NULL values in the input column are skipped and do not yield a result row.

The following example shows UNPIVOT with INCLUDE NULLS.

```
SELECT *
FROM (
 SELECT red, green, blue
 FROM count_by_color
) UNPIVOT INCLUDE NULLS (
 cnt FOR color IN (red, green, blue)
);
```

The following is the resulting output.

| color | cnt |
|-------|-----|
| red   | 15  |
| red   | 35  |
| red   | 10  |
| green | 20  |
| green |     |

|       |  |    |
|-------|--|----|
| green |  | 23 |
| blue  |  | 7  |
| blue  |  | 40 |
| blue  |  |    |

If the INCLUDING NULLS parameter is set, NULL input values generate result rows.

The following query shows UNPIVOT with quality as an implicit column.

```
SELECT *
FROM count_by_color UNPIVOT (
 cnt FOR color IN (red, green, blue)
);
```

The query results in the following output.

| quality |  | color |  | cnt |
|---------|--|-------|--|-----|
| high    |  | red   |  | 15  |
| normal  |  | red   |  | 35  |
| low     |  | red   |  | 10  |
| high    |  | green |  | 20  |
| low     |  | green |  | 23  |
| high    |  | blue  |  | 7   |
| normal  |  | blue  |  | 40  |

Columns of the input table that are not referenced in the UNPIVOT definition are added implicitly to the result table. In the example, this is the case for the quality column.

The following example shows UNPIVOT with aliases for values in the IN list.

```
SELECT *
FROM count_by_color UNPIVOT (
 cnt FOR color IN (red AS r, green AS g, blue AS b)
);
```

The previous query results in the following output.

| quality |  | color |  | cnt |
|---------|--|-------|--|-----|
| high    |  | r     |  | 15  |
| normal  |  | r     |  | 35  |
| low     |  | r     |  | 10  |
| high    |  | g     |  | 20  |
| low     |  | g     |  | 23  |
| high    |  | b     |  | 7   |
| normal  |  | b     |  | 40  |

The UNPIVOT operator accepts optional aliases on each IN list value. Each alias provides customization of the data in each value column.

The following are usage notes for UNPIVOT.

- UNPIVOT can be applied to tables, sub-queries, and common table expressions (CTEs). UNPIVOT cannot be applied to any JOIN expressions, recursive CTEs, PIVOT, or UNPIVOT expressions. Also not supported are SUPER unnested expressions and Redshift Spectrum nested tables.
- The UNPIVOT IN list must contain only input table column references. The IN list columns must have a common type that they are all compatible with. The UNPIVOT value column has this common type. The UNPIVOT name column is of type VARCHAR.

- If an IN list value does not have an alias, UNPIVOT uses the column name as a default value.

## JOIN examples

A SQL JOIN clause is used to combine the data from two or more tables based on common fields. The results might or might not change depending on the join method specified. For more information about the syntax of a JOIN clause, see [Parameters \(p. 938\)](#).

The following examples use data from the TICKIT sample data. For more information about the database schema, see [Sample database \(p. 1704\)](#). To learn how to load sample data, see [Using a sample dataset](#) in the *Amazon Redshift Getting Started Guide*.

The following query is an inner join (without the JOIN keyword) between the LISTING table and SALES table, where the LISTID from the LISTING table is between 1 and 5. This query matches LISTID column values in the LISTING table (the left table) and SALES table (the right table). The results show that LISTID 1, 4, and 5 match the criteria.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing, sales
where listing.listid = sales.listid
and listing.listid between 1 and 5
group by 1
order by 1;

listid | price | comm
-----+-----+-----
 1 | 728.00 | 109.20
 4 | 76.00 | 11.40
 5 | 525.00 | 78.75
```

The following query is a left outer join. Left and right outer joins retain values from one of the joined tables when no match is found in the other table. The left and right tables are the first and second tables listed in the syntax. NULL values are used to fill the "gaps" in the result set. This query matches LISTID column values in the LISTING table (the left table) and the SALES table (the right table). The results show that LISTIDs 2 and 3 did not result in any sales.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing left outer join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price | comm
-----+-----+-----
 1 | 728.00 | 109.20
 2 | NULL | NULL
 3 | NULL | NULL
 4 | 76.00 | 11.40
 5 | 525.00 | 78.75
```

The following query is a right outer join. This query matches LISTID column values in the LISTING table (the left table) and the SALES table (the right table). The results show that LISTIDs 1, 4, and 5 match the criteria.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing right outer join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
group by 1
order by 1;
```

| listid | price  | comm   |
|--------|--------|--------|
| 1      | 728.00 | 109.20 |
| 4      | 76.00  | 11.40  |
| 5      | 525.00 | 78.75  |

The following query is a full join. Full joins retain values from the joined tables when no match is found in the other table. The left and right tables are the first and second tables listed in the syntax. NULL values are used to fill the "gaps" in the result set. This query matches LISTID column values in the LISTING table (the left table) and the SALES table (the right table). The results show that LISTIDs 2 and 3 did not result in any sales.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing full join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price | comm
-----+-----+
1 | 728.00 | 109.20
2 | NULL | NULL
3 | NULL | NULL
4 | 76.00 | 11.40
5 | 525.00 | 78.75
```

The following query is a full join. This query matches LISTID column values in the LISTING table (the left table) and the SALES table (the right table). Only rows that do not result in any sales (LISTIDs 2 and 3) are in the results.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing full join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
and (listing.listid IS NULL or sales.listid IS NULL)
group by 1
order by 1;

listid | price | comm
-----+-----+
2 | NULL | NULL
3 | NULL | NULL
```

The following example is an inner join with the ON clause. In this case, NULL rows are not returned.

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from sales join listing
on sales.listid=listing.listid and sales.eventid=listing.eventid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price | comm
-----+-----+
1 | 728.00 | 109.20
4 | 76.00 | 11.40
5 | 525.00 | 78.75
```

The following query is a cross join or Cartesian join of the LISTING table and the SALES table with a predicate to limit the results. This query matches LISTID column values in the SALES table and the

LISTING table for LISTIDs 1, 2, 3, 4, and 5 in both tables. The results show that 20 rows match the criteria.

```
select sales.listid as sales_listid, listing.listid as listing_listid
from sales cross join listing
where sales.listid between 1 and 5
and listing.listid between 1 and 5
order by 1,2;

sales_listid | listing_listid
-----+-----
1 | 1
1 | 2
1 | 3
1 | 4
1 | 5
4 | 1
4 | 2
4 | 3
4 | 4
4 | 5
5 | 1
5 | 2
5 | 3
5 | 4
5 | 5
5 | 5
```

The following example is a natural join between two tables. In this case, the columns listid, sellerid, eventid, and dateid have identical names and data types in both tables and so are used as the join columns. The results are limited to five rows.

```
select listid, sellerid, eventid, dateid, numtickets
from listing natural join sales
order by 1
limit 5;

listid | sellerid | eventid | dateid | numtickets
-----+-----+-----+-----+-----
113 | 29704 | 4699 | 2075 | 22
115 | 39115 | 3513 | 2062 | 14
116 | 43314 | 8675 | 1910 | 28
118 | 6079 | 1611 | 1862 | 9
163 | 24880 | 8253 | 1888 | 14
```

The following example is a join between two tables with the USING clause. In this case, the columns listid and eventid are used as the join columns. The results are limited to five rows.

```
select listid, listing.sellerid, eventid, listing.dateid, numtickets
from listing join sales
using (listid, eventid)
order by 1
limit 5;

listid | sellerid | eventid | dateid | numtickets
-----+-----+-----+-----+-----
1 | 36861 | 7872 | 1850 | 10
4 | 8117 | 4337 | 1970 | 8
```

|   |  |       |  |      |  |      |  |    |
|---|--|-------|--|------|--|------|--|----|
| 5 |  | 1616  |  | 8647 |  | 1963 |  | 4  |
| 5 |  | 1616  |  | 8647 |  | 1963 |  | 4  |
| 6 |  | 47402 |  | 8240 |  | 2053 |  | 18 |

The following query is an inner join of two subqueries in the FROM clause. The query finds the number of sold and unsold tickets for different categories of events (concerts and shows). The FROM clause subqueries are *table* subqueries; they can return multiple columns and rows.

```
select catgroup1, sold, unsold
from
(select catgroup, sum(qtysold) as sold
from category c, event e, sales s
where c.catid = e.catid and e.eventid = s.eventid
group by catgroup) as a(catgroup1, sold)
join
(select catgroup, sum(numtickets)-sum(qtysold) as unsold
from category c, event e, sales s, listing l
where c.catid = e.catid and e.eventid = s.eventid
and s.listid = l.listid
group by catgroup) as b(catgroup2, unsold)

on a.catgroup1 = b.catgroup2
order by 1;

catgroup1 | sold | unsold
-----+-----+
Concerts | 195444 | 1067199
Shows | 149905 | 817736
```

## WHERE clause

The WHERE clause contains conditions that either join tables or apply predicates to columns in tables. Tables can be inner-joined by using appropriate syntax in either the WHERE clause or the FROM clause. Outer join criteria must be specified in the FROM clause.

### Syntax

```
[WHERE condition]
```

### *condition*

Any search condition with a Boolean result, such as a join condition or a predicate on a table column. The following examples are valid join conditions:

```
sales.listid=listing.listid
sales.listid<>listing.listid
```

The following examples are valid conditions on columns in tables:

```
catgroup like 'S%'
venueseats between 20000 and 50000
eventname in('Jersey Boys', 'Spamalot')
year=2008
length(catdesc)>25
date_part(month, caldate)=6
```

Conditions can be simple or complex; for complex conditions, you can use parentheses to isolate logical units. In the following example, the join condition is enclosed by parentheses.

```
where (category.catid=event.catid) and category.catid in(6,7,8)
```

## Usage notes

You can use aliases in the WHERE clause to reference select list expressions.

You can't restrict the results of aggregate functions in the WHERE clause; use the HAVING clause for this purpose.

Columns that are restricted in the WHERE clause must derive from table references in the FROM clause.

## Example

The following query uses a combination of different WHERE clause restrictions, including a join condition for the SALES and EVENT tables, a predicate on the EVENTNAME column, and two predicates on the STARTTIME column.

```
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Hannah Montana'
and date_part(quarter, starttime) in(1,2)
and date_part(year, starttime) = 2008
order by 3 desc, 4, 2, 1 limit 10;

eventname | starttime | costperticket | qtysold
-----+-----+-----+-----
Hannah Montana | 2008-06-07 14:00:00 | 1706.00000000 | 2
Hannah Montana | 2008-05-01 19:00:00 | 1658.00000000 | 2
Hannah Montana | 2008-06-07 14:00:00 | 1479.00000000 | 1
Hannah Montana | 2008-06-07 14:00:00 | 1479.00000000 | 3
Hannah Montana | 2008-06-07 14:00:00 | 1163.00000000 | 1
Hannah Montana | 2008-06-07 14:00:00 | 1163.00000000 | 2
Hannah Montana | 2008-06-07 14:00:00 | 1163.00000000 | 4
Hannah Montana | 2008-05-01 19:00:00 | 497.00000000 | 1
Hannah Montana | 2008-05-01 19:00:00 | 497.00000000 | 2
Hannah Montana | 2008-05-01 19:00:00 | 497.00000000 | 4
(10 rows)
```

## Oracle-Style outer joins in the WHERE clause

For Oracle compatibility, Amazon Redshift supports the Oracle outer-join operator (+) in WHERE clause join conditions. This operator is intended for use only in defining outer-join conditions; don't try to use it in other contexts. Other uses of this operator are silently ignored in most cases.

An outer join returns all of the rows that the equivalent inner join would return, plus non-matching rows from one or both tables. In the FROM clause, you can specify left, right, and full outer joins. In the WHERE clause, you can specify left and right outer joins only.

To outer join tables TABLE1 and TABLE2 and return non-matching rows from TABLE1 (a left outer join), specify TABLE1 LEFT OUTER JOIN TABLE2 in the FROM clause or apply the (+) operator to all joining columns from TABLE2 in the WHERE clause. For all rows in TABLE1 that have no matching rows in TABLE2, the result of the query contains nulls for any select list expressions that contain columns from TABLE2.

To produce the same behavior for all rows in TABLE2 that have no matching rows in TABLE1, specify TABLE1 RIGHT OUTER JOIN TABLE2 in the FROM clause or apply the (+) operator to all joining columns from TABLE1 in the WHERE clause.

## Basic syntax

```
[WHERE {
[table1.column1 = table2.column1(+)]
[table1.column1(+) = table2.column1]
}
```

The first condition is equivalent to:

```
from table1 left outer join table2
on table1.column1=table2.column1
```

The second condition is equivalent to:

```
from table1 right outer join table2
on table1.column1=table2.column1
```

### Note

The syntax shown here covers the simple case of an equijoin over one pair of joining columns. However, other types of comparison conditions and multiple pairs of joining columns are also valid.

For example, the following WHERE clause defines an outer join over two pairs of columns. The (+) operator must be attached to the same table in both conditions:

```
where table1.col1 > table2.col1(+)
and table1.col2 = table2.col2(+)
```

## Usage notes

Where possible, use the standard FROM clause OUTER JOIN syntax instead of the (+) operator in the WHERE clause. Queries that contain the (+) operator are subject to the following rules:

- You can only use the (+) operator in the WHERE clause, and only in reference to columns from tables or views.
- You can't apply the (+) operator to expressions. However, an expression can contain columns that use the (+) operator. For example, the following join condition returns a syntax error:

```
event.eventid*10(+) = category.catid
```

However, the following join condition is valid:

```
event.eventid(+) * 10 = category.catid
```

- You can't use the (+) operator in a query block that also contains FROM clause join syntax.
- If two tables are joined over multiple join conditions, you must use the (+) operator in all or none of these conditions. A join with mixed syntax styles runs as an inner join, without warning.
- The (+) operator doesn't produce an outer join if you join a table in the outer query with a table that results from an inner query.
- To use the (+) operator to outer-join a table to itself, you must define table aliases in the FROM clause and reference them in the join condition:

```
select count(*)
```

```
from event a, event b
where a.eventid(+) = b.catid;

count

8798
(1 row)
```

- You can't combine a join condition that contains the (+) operator with an OR condition or an IN condition. For example:

```
select count(*) from sales, listing
where sales.listid(+) = listing.listid or sales.salesid=0;
ERROR: Outer join operator (+) not allowed in operand of OR or IN.
```

- In a WHERE clause that outer-joins more than two tables, the (+) operator can be applied only once to a given table. In the following example, the SALES table can't be referenced with the (+) operator in two successive joins.

```
select count(*) from sales, listing, event
where sales.listid(+) = listing.listid and sales.dateid(+) = date.dateid;
ERROR: A table may be outer joined to at most one other table.
```

- If the WHERE clause outer-join condition compares a column from TABLE2 with a constant, apply the (+) operator to the column. If you don't include the operator, the outer-joined rows from TABLE1, which contain nulls for the restricted column, are eliminated. See the Examples section below.

## Examples

The following join query specifies a left outer join of the SALES and LISTING tables over their LISTID columns:

```
select count(*)
from sales, listing
where sales.listid = listing.listid(+);

count

172456
(1 row)
```

The following equivalent query produces the same result but uses FROM clause join syntax:

```
select count(*)
from sales left outer join listing on sales.listid = listing.listid;

count

172456
(1 row)
```

The SALES table doesn't contain records for all listings in the LISTING table because not all listings result in sales. The following query outer-joins SALES and LISTING and returns rows from LISTING even when the SALES table reports no sales for a given list ID. The PRICE and COMM columns, derived from the SALES table, contain nulls in the result set for those non-matching rows.

```
select listing.listid, sum(pricepaid) as price,
sum(commission) as comm
from listing, sales
```

```
where sales.listid(+) = listing.listid and listing.listid between 1 and 5
group by 1 order by 1;

listid | price | comm
-----+-----+-----
1 | 728.00 | 109.20
2 |
3 |
4 | 76.00 | 11.40
5 | 525.00 | 78.75
(5 rows)
```

Note that when the WHERE clause join operator is used, the order of the tables in the FROM clause doesn't matter.

An example of a more complex outer join condition in the WHERE clause is the case where the condition consists of a comparison between two table columns *and* a comparison with a constant:

```
where category.catid=event.catid(+) and eventid(+)=796;
```

Note that the (+) operator is used in two places: first in the equality comparison between the tables and second in the comparison condition for the EVENTID column. The result of this syntax is the preservation of the outer-joined rows when the restriction on EVENTID is evaluated. If you remove the (+) operator from the EVENTID restriction, the query treats this restriction as a filter, not as part of the outer-join condition. In turn, the outer-joined rows that contain nulls for EVENTID are eliminated from the result set.

Here is a complete query that illustrates this behavior:

```
select catname, catgroup, eventid
from category, event
where category.catid=event.catid(+) and eventid(+)=796;

catname | catgroup | eventid
-----+-----+-----
Classical | Concerts |
Jazz | Concerts |
MLB | Sports |
MLS | Sports |
Musicals | Shows | 796
NBA | Sports |
NFL | Sports |
NHL | Sports |
Opera | Shows |
Plays | Shows |
Pop | Concerts |
(11 rows)
```

The equivalent query using FROM clause syntax is as follows:

```
select catname, catgroup, eventid
from category left join event
on category.catid=event.catid and eventid=796;
```

If you remove the second (+) operator from the WHERE clause version of this query, it returns only 1 row (the row where eventid=796).

```
select catname, catgroup, eventid
from category, event
```

```
where category.catid=event.catid(+) and eventid=796;
catname | catgroup | eventid
-----+-----+-----
Musicals | Shows | 796
(1 row)
```

## GROUP BY clause

The GROUP BY clause identifies the grouping columns for the query. Grouping columns must be declared when the query computes aggregates with standard functions such as SUM, AVG, and COUNT. For more information, see [Aggregate functions \(p. 1022\)](#).

### Syntax

```
GROUP BY group_by_clause [, ...]

group_by_clause := {
 expr |
 GROUPING SETS (() | group_by_clause [, ...]) |
 ROLLUP (expr [, ...]) |
 CUBE (expr [, ...])
}
```

### Parameters

#### *expr*

The list of columns or expressions must match the list of non-aggregate expressions in the select list of the query. For example, consider the following simple query.

```
select listid, eventid, sum(pricepaid) as revenue,
count(qtysold) as numtix
from sales
group by listid, eventid
order by 3, 4, 2, 1
limit 5;

listid | eventid | revenue | numtix
-----+-----+-----+-----
89397 | 47 | 20.00 | 1
106590 | 76 | 20.00 | 1
124683 | 393 | 20.00 | 1
103037 | 403 | 20.00 | 1
147685 | 429 | 20.00 | 1
(5 rows)
```

In this query, the select list consists of two aggregate expressions. The first uses the SUM function and the second uses the COUNT function. The remaining two columns, LISTID and EVENTID, must be declared as grouping columns.

Expressions in the GROUP BY clause can also reference the select list by using ordinal numbers. For example, the previous example could be abbreviated as follows.

```
select listid, eventid, sum(pricepaid) as revenue,
count(qtysold) as numtix
from sales
group by 1,2
order by 3, 4, 2, 1
```

```
limit 5;

listid | eventid | revenue | numtix
-----+-----+-----+-----
89397 | 47 | 20.00 | 1
106590 | 76 | 20.00 | 1
124683 | 393 | 20.00 | 1
103037 | 403 | 20.00 | 1
147685 | 429 | 20.00 | 1
(5 rows)
```

#### *GROUPING SETS/ROLLUP/CUBE*

You can use the aggregation extensions GROUPING SETS, ROLLUP, and CUBE to perform the work of multiple GROUP BY operations in a single statement. You can only use aggregation extensions on preview clusters or preview serverless workgroups. For more information on aggregation extensions and related functions, see [Aggregation extensions \(p. 955\)](#).

### Aggregation extensions

Amazon Redshift supports aggregation extensions to do the work of multiple GROUP BY operations in a single statement.

The examples for aggregation extensions use the `orders` table, which holds sales data for an electronics company. You can create `orders` with the following.

```
CREATE TABLE ORDERS (
 ID INT,
 PRODUCT CHAR(20),
 CATEGORY CHAR(20),
 PRE_OWNED CHAR(1),
 COST DECIMAL
);

INSERT INTO ORDERS VALUES
 (0, 'laptop', 'computers', 'T', 1000),
 (1, 'smartphone', 'cellphones', 'T', 800),
 (2, 'smartphone', 'cellphones', 'T', 810),
 (3, 'laptop', 'computers', 'F', 1050),
 (4, 'mouse', 'computers', 'F', 50);
```

#### *GROUPING SETS*

Computes one or more grouping sets in a single statement. A grouping set is the set of a single GROUP BY clause, a set of 0 or more columns by which you can group a query's result set. GROUP BY GROUPING SETS is equivalent to running a UNION ALL query on one result set grouped by different columns. For example, GROUP BY GROUPING SETS((a), (b)) is equivalent to GROUP BY a UNION ALL GROUP BY b.

The following example returns the cost of the order table's products grouped according to both the products' categories and the kind of products sold.

```
SELECT category, product, sum(cost) as total
FROM orders
GROUP BY GROUPING SETS(category, product);

category | product | total
-----+-----+-----
computers | | 2100
cellphones | | 1610
```

|          |            |      |
|----------|------------|------|
|          | laptop     | 2050 |
|          | smartphone | 1610 |
|          | mouse      | 50   |
| (5 rows) |            |      |

### ROLLUP

Assumes a hierarchy where preceding columns are considered the parents of subsequent columns. ROLLUP groups data by the provided columns, returning extra subtotal rows representing the totals throughout all levels of grouping columns, in addition to the grouped rows. For example, you can use GROUP BY ROLLUP((a), (b)) to return a result set grouped first by a, then by b while assuming that b is a subsection of a. ROLLUP also returns a row with the whole result set without grouping columns.

GROUP BY ROLLUP((a), (b)) is equivalent to GROUP BY GROUPING SETS((a,b), (a), ()).

The following example returns the cost of the order table's products grouped first by category and then product, with product as a subdivision of category.

```
SELECT category, product, sum(cost) as total
FROM orders
GROUP BY ROLLUP(category, product) ORDER BY 1,2;
```

| category   | product    | total |
|------------|------------|-------|
| cellphones | smartphone | 1610  |
| cellphones |            | 1610  |
| computers  | laptop     | 2050  |
| computers  | mouse      | 50    |
| computers  |            | 2100  |
|            |            | 3710  |
| (6 rows)   |            |       |

### CUBE

Groups data by the provided columns, returning extra subtotal rows representing the totals throughout all levels of grouping columns, in addition to the grouped rows. CUBE returns the same rows as ROLLUP, while adding additional subtotal rows for every combination of grouping column not covered by ROLLUP. For example, you can use GROUP BY CUBE ((a), (b)) to return a result set grouped first by a, then by b while assuming that b is a subsection of a, then by b alone. CUBE also returns a row with the whole result set without grouping columns.

GROUP BY CUBE((a), (b)) is equivalent to GROUP BY GROUPING SETS((a, b), (a), (b), ()).

The following example returns the cost of the order table's products grouped first by category and then product, with product as a subdivision of category. Unlike the preceding example for ROLLUP, the statement returns results for every combination of grouping column.

```
SELECT category, product, sum(cost) as total
FROM orders
GROUP BY CUBE(category, product) ORDER BY 1,2;
```

| category   | product    | total |
|------------|------------|-------|
| cellphones | smartphone | 1610  |
| cellphones |            | 1610  |
| computers  | laptop     | 2050  |
| computers  | mouse      | 50    |
| computers  |            | 2100  |
|            | laptop     | 2050  |
| (6 rows)   |            |       |

|          |            |      |
|----------|------------|------|
| (9 rows) |            |      |
|          | mouse      | 50   |
|          | smartphone | 1610 |
|          |            | 3710 |

### *GROUPING/GROUPING\_ID functions*

ROLLUP and CUBE add NULL values to the result set to indicate subtotal rows. For example, GROUP BY ROLLUP((a), (b)) returns one or more rows that have a value of NULL in the b grouping column to indicate they are subtotals of fields in the a grouping column. These NULL values serve only to satisfy the format of returning tuples.

When you run GROUP BY operations with ROLLUP and CUBE on relations that store NULL values themselves, this can produce result sets with rows that appear to have identical grouping columns. Returning to the previous example, if the b grouping column contains a stored NULL value, GROUP BY ROLLUP((a), (b)) returns a row with a value of NULL in the b grouping column that isn't a subtotal.

To distinguish between NULL values created by ROLLUP and CUBE, and the NULL values stored in the tables themselves, you can use the GROUPING function, or its alias GROUPING\_ID. GROUPING takes a single grouping set as its argument, and for each row in the result set returns a 0 or 1 bit value corresponding to the grouping column in that position, and then converts that value into an integer. If the value in that position is a NULL value created by an aggregation extension, GROUPING returns 1. It returns 0 for all other values, including stored NULL values.

For example, GROUPING(category, product) can return the following values for a given row, depending on the grouping column values for that row. For the purposes of this example, all NULL values in the table are NULL values created by an aggregation extension.

| category column | product column | GROUPING function bit value | Decimal value |
|-----------------|----------------|-----------------------------|---------------|
| not NULL        | not NULL       | 00                          | 0             |
| not NULL        | NULL           | 01                          | 1             |
| NULL            | not NULL       | 10                          | 2             |
| NULL            | NULL           | 11                          | 3             |

GROUPING functions appear in the SELECT list portion of the query in the following format.

```
SELECT ... [GROUPING(expr)...] ...
 GROUP BY ... {CUBE | ROLLUP| GROUPING SETS} (expr) ...
```

The following example is the same as the preceding example for CUBE, but with the addition of GROUPING functions for its grouping sets.

```
SELECT category, product,
 GROUPING(category) as grouping0,
 GROUPING(product) as grouping1,
 GROUPING(category, product) as grouping2,
 sum(cost) as total
 FROM orders
 GROUP BY CUBE(category, product) ORDER BY 3,1,2;

 category | product | grouping0 | grouping1 | grouping2 | total
```

|            |            |  | 0 | 0 | 0 | 1610 |
|------------|------------|--|---|---|---|------|
| cellphones | smartphone |  | 0 | 0 | 0 | 1610 |
| cellphones |            |  | 0 | 1 | 1 | 1610 |
| computers  | laptop     |  | 0 | 0 | 0 | 2050 |
| computers  | mouse      |  | 0 | 0 | 0 | 50   |
| computers  |            |  | 0 | 1 | 1 | 2100 |
|            | laptop     |  | 1 | 0 | 2 | 2050 |
|            | mouse      |  | 1 | 0 | 2 | 50   |
|            | smartphone |  | 1 | 0 | 2 | 1610 |
|            |            |  | 1 | 1 | 3 | 3710 |

(9 rows)

### Partial ROLLUP and CUBE

You can run ROLLUP and CUBE operations with only a portion of the subtotals.

The syntax for partial ROLLUP and CUBE operations is as follows.

```
GROUP BY expr1, { ROLLUP | CUBE }(expr2, [, ...])
```

Here, the GROUP BY clause only creates subtotal rows at the level of *expr2* and onwards.

The following examples show partial ROLLUP and CUBE operations on the orders table, grouping first by whether a product is pre-owned and then running ROLLUP and CUBE on the category and product columns.

| SELECT pre_owned, category, product,<br>GROUPING(category, product, pre_owned) as group_id,<br>sum(cost) as total<br>FROM orders<br>GROUP BY pre_owned, ROLLUP(category, product) ORDER BY 4,1,2,3; |            |            |          |       |  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|------------|----------|-------|--|
| pre_owned                                                                                                                                                                                           | category   | product    | group_id | total |  |
| F                                                                                                                                                                                                   | computers  | laptop     | 0        | 1050  |  |
| F                                                                                                                                                                                                   | computers  | mouse      | 0        | 50    |  |
| T                                                                                                                                                                                                   | cellphones | smartphone | 0        | 1610  |  |
| T                                                                                                                                                                                                   | computers  | laptop     | 0        | 1000  |  |
| F                                                                                                                                                                                                   | computers  |            | 2        | 1100  |  |
| T                                                                                                                                                                                                   | cellphones |            | 2        | 1610  |  |
| T                                                                                                                                                                                                   | computers  |            | 2        | 1000  |  |
| F                                                                                                                                                                                                   |            |            | 6        | 1100  |  |
| T                                                                                                                                                                                                   |            |            | 6        | 2610  |  |

(9 rows)

| SELECT pre_owned, category, product,<br>GROUPING(category, product, pre_owned) as group_id,<br>sum(cost) as total<br>FROM orders<br>GROUP BY pre_owned, CUBE(category, product) ORDER BY 4,1,2,3; |            |            |          |       |  |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|------------|----------|-------|--|
| pre_owned                                                                                                                                                                                         | category   | product    | group_id | total |  |
| F                                                                                                                                                                                                 | computers  | laptop     | 0        | 1050  |  |
| F                                                                                                                                                                                                 | computers  | mouse      | 0        | 50    |  |
| T                                                                                                                                                                                                 | cellphones | smartphone | 0        | 1610  |  |
| T                                                                                                                                                                                                 | computers  | laptop     | 0        | 1000  |  |
| F                                                                                                                                                                                                 | computers  |            | 2        | 1100  |  |
| T                                                                                                                                                                                                 | cellphones |            | 2        | 1610  |  |
| T                                                                                                                                                                                                 | computers  |            | 2        | 1000  |  |
| F                                                                                                                                                                                                 |            | laptop     | 4        | 1050  |  |
| F                                                                                                                                                                                                 |            | mouse      | 4        | 50    |  |

|           |  |  |            |  |   |      |
|-----------|--|--|------------|--|---|------|
| T         |  |  | laptop     |  | 4 | 1000 |
| T         |  |  | smartphone |  | 4 | 1610 |
| F         |  |  |            |  | 6 | 1100 |
| T         |  |  |            |  | 6 | 2610 |
| (13 rows) |  |  |            |  |   |      |

Since the pre-owned column isn't included in the ROLLUP and CUBE operations, there's no grand total row that includes all other rows.

### Concatenated grouping

You can concatenate multiple GROUPING SETS/ROLLUP/CUBE clauses to calculate different levels of subtotals. Concatenated groupings return the Cartesian product of the provided grouping sets.

The syntax for concatenating GROUPING SETS/ROLLUP/CUBE clauses is as follows.

```
GROUP BY {ROLLUP|CUBE|GROUPING SETS}(expr1[, ...]),
 {ROLLUP|CUBE|GROUPING SETS}(expr1[, ...])[, ...]
```

Consider the following example to see how a small concatenated grouping can produce a large final result set.

| pre_owned | category   | product    | group_id | total |
|-----------|------------|------------|----------|-------|
| F         | computers  | laptop     | 0        | 1050  |
| F         | computers  | mouse      | 0        | 50    |
| T         | cellphones | smartphone | 0        | 1610  |
| T         | computers  | laptop     | 0        | 1000  |
|           | cellphones | smartphone | 1        | 1610  |
|           | computers  | laptop     | 1        | 2050  |
|           | computers  | mouse      | 1        | 50    |
| F         | computers  |            | 2        | 1100  |
| T         | cellphones |            | 2        | 1610  |
| T         | computers  |            | 2        | 1000  |
|           | cellphones |            | 3        | 1610  |
|           | computers  |            | 3        | 2100  |
| F         |            | laptop     | 4        | 1050  |
| F         |            | mouse      | 4        | 50    |
| T         |            | laptop     | 4        | 1000  |
| T         |            | smartphone | 4        | 1610  |
|           |            | laptop     | 5        | 2050  |
|           |            | mouse      | 5        | 50    |
|           |            | smartphone | 5        | 1610  |
| F         |            |            | 6        | 1100  |
| T         |            |            | 6        | 2610  |
|           |            |            | 7        | 3710  |
| (22 rows) |            |            |          |       |

### Nested grouping

You can use GROUPING SETS/ROLLUP/CUBE operations as your GROUPING SETS *expr* to form a nested grouping. The sub grouping inside nested GROUPING SETS is flattened.

The syntax for nested grouping is as follows.

```
GROUP BY GROUPING SETS({ROLLUP|CUBE|GROUPING SETS}(expr[, ...])[, ...])
```

Consider the following example.

```
SELECT category, product, pre_owned,
 GROUPING(category, product, pre_owned) as group_id,
 sum(cost) as total
 FROM orders
 GROUP BY GROUPING SETS(ROLLUP(category), CUBE(product, pre_owned))
 ORDER BY 4,1,2,3;
```

| category   | product    | pre_owned | group_id | total |
|------------|------------|-----------|----------|-------|
| cellphones |            |           | 3        | 1610  |
| computers  |            |           | 3        | 2100  |
|            | laptop     | F         | 4        | 1050  |
|            | laptop     | T         | 4        | 1000  |
|            | mouse      | F         | 4        | 50    |
|            | smartphone | T         | 4        | 1610  |
|            | laptop     |           | 5        | 2050  |
|            | mouse      |           | 5        | 50    |
|            | smartphone |           | 5        | 1610  |
|            |            | F         | 6        | 1100  |
|            |            | T         | 6        | 2610  |
|            |            |           | 7        | 3710  |
|            |            |           | 7        | 3710  |

(13 rows)

Note that because both ROLLUP(category) and CUBE(product, pre\_owned) contain the grouping set (), the row representing the grand total is duplicated.

#### Usage notes

- The GROUP BY clause supports up to 64 grouping sets. In the case of ROLLUP and CUBE, or some combination of GROUPING SETS, ROLLUP, and CUBE, this limitation applies to the implied number of grouping sets. For example, GROUP BY CUBE((a), (b)) counts as 4 grouping sets, not 2.
- You can't use constants as grouping columns when using aggregation extensions.
- You can't make a grouping set that contains duplicate columns.

## HAVING clause

The HAVING clause applies a condition to the intermediate grouped result set that a query returns.

#### Syntax

```
[HAVING condition]
```

For example, you can restrict the results of a SUM function:

```
having sum(pricepaid) >10000
```

The HAVING condition is applied after all WHERE clause conditions are applied and GROUP BY operations are completed.

The condition itself takes the same form as any WHERE clause condition.

## Usage notes

- Any column that is referenced in a HAVING clause condition must be either a grouping column or a column that refers to the result of an aggregate function.
- In a HAVING clause, you can't specify:
  - An alias that was defined in the select list. You must repeat the original, unaliased expression.
  - An ordinal number that refers to a select list item. Only the GROUP BY and ORDER BY clauses accept ordinal numbers.

## Examples

The following query calculates total ticket sales for all events by name, then eliminates events where the total sales were less than \$800,000. The HAVING condition is applied to the results of the aggregate function in the select list: sum(pricepaid).

```
select eventname, sum(pricepaid)
from sales join event on sales.eventid = event.eventid
group by 1
having sum(pricepaid) > 800000
order by 2 desc, 1;

eventname | sum
-----+-----
Mamma Mia! | 1135454.00
Spring Awakening | 972855.00
The Country Girl | 910563.00
Macbeth | 862580.00
Jersey Boys | 811877.00
Legally Blonde | 804583.00
(6 rows)
```

The following query calculates a similar result set. In this case, however, the HAVING condition is applied to an aggregate that isn't specified in the select list: sum(qtysold). Events that did not sell more than 2,000 tickets are eliminated from the final result.

```
select eventname, sum(pricepaid)
from sales join event on sales.eventid = event.eventid
group by 1
having sum(qtysold) >2000
order by 2 desc, 1;

eventname | sum
-----+-----
Mamma Mia! | 1135454.00
Spring Awakening | 972855.00
The Country Girl | 910563.00
Macbeth | 862580.00
Jersey Boys | 811877.00
Legally Blonde | 804583.00
Chicago | 790993.00
Spamalot | 714307.00
(8 rows)
```

## UNION, INTERSECT, and EXCEPT

### Topics

- [Syntax \(p. 962\)](#)

- [Parameters \(p. 962\)](#)
- [Order of evaluation for set operators \(p. 963\)](#)
- [Usage notes \(p. 963\)](#)
- [Example UNION queries \(p. 964\)](#)
- [Example UNION ALL query \(p. 966\)](#)
- [Example INTERSECT queries \(p. 967\)](#)
- [Example EXCEPT query \(p. 968\)](#)

The UNION, INTERSECT, and EXCEPT *set operators* are used to compare and merge the results of two separate query expressions. For example, if you want to know which users of a website are both buyers and sellers but their user names are stored in separate columns or tables, you can find the *intersection* of these two types of users. If you want to know which website users are buyers but not sellers, you can use the EXCEPT operator to find the *difference* between the two lists of users. If you want to build a list of all users, regardless of role, you can use the UNION operator.

## Syntax

```
query
{ UNION [ALL] | INTERSECT | EXCEPT | MINUS }
query
```

## Parameters

*query*

A query expression that corresponds, in the form of its select list, to a second query expression that follows the UNION, INTERSECT, or EXCEPT operator. The two expressions must contain the same number of output columns with compatible data types; otherwise, the two result sets can't be compared and merged. Set operations don't allow implicit conversion between different categories of data types; for more information, see [Type compatibility and conversion \(p. 597\)](#).

You can build queries that contain an unlimited number of query expressions and link them with UNION, INTERSECT, and EXCEPT operators in any combination. For example, the following query structure is valid, assuming that the tables T1, T2, and T3 contain compatible sets of columns:

```
select * from t1
union
select * from t2
except
select * from t3
order by c1;
```

## UNION

Set operation that returns rows from two query expressions, regardless of whether the rows derive from one or both expressions.

## INTERSECT

Set operation that returns rows that derive from two query expressions. Rows that aren't returned by both expressions are discarded.

## EXCEPT | MINUS

Set operation that returns rows that derive from one of two query expressions. To qualify for the result, rows must exist in the first result table but not the second. MINUS and EXCEPT are exact synonyms.

## ALL

The ALL keyword retains any duplicate rows that are produced by UNION. The default behavior when the ALL keyword isn't used is to discard these duplicates. INTERSECT ALL, EXCEPT ALL, and MINUS ALL aren't supported.

## Order of evaluation for set operators

The UNION and EXCEPT set operators are left-associative. If parentheses aren't specified to influence the order of precedence, a combination of these set operators is evaluated from left to right. For example, in the following query, the UNION of T1 and T2 is evaluated first, then the EXCEPT operation is performed on the UNION result:

```
select * from t1
union
select * from t2
except
select * from t3
order by c1;
```

The INTERSECT operator takes precedence over the UNION and EXCEPT operators when a combination of operators is used in the same query. For example, the following query evaluates the intersection of T2 and T3, then union the result with T1:

```
select * from t1
union
select * from t2
intersect
select * from t3
order by c1;
```

By adding parentheses, you can enforce a different order of evaluation. In the following case, the result of the union of T1 and T2 is intersected with T3, and the query is likely to produce a different result.

```
(select * from t1
union
select * from t2)
intersect
(select * from t3)
order by c1;
```

## Usage notes

- The column names returned in the result of a set operation query are the column names (or aliases) from the tables in the first query expression. Because these column names are potentially misleading, in that the values in the column derive from tables on either side of the set operator, you might want to provide meaningful aliases for the result set.
- A query expression that precedes a set operator should not contain an ORDER BY clause. An ORDER BY clause produces meaningful sorted results only when it is used at the end of a query that contains set operators. In this case, the ORDER BY clause applies to the final results of all of the set operations. The outermost query can also contain standard LIMIT and OFFSET clauses.
- The LIMIT and OFFSET clauses aren't supported as a means of restricting the number of rows returned by an intermediate result of a set operation. For example, the following query returns an error:

```
(select listid from listing
```

```
limit 10)
intersect
select listid from sales;
ERROR: LIMIT may not be used within input to set operations.
```

- When set operator queries return decimal results, the corresponding result columns are promoted to return the same precision and scale. For example, in the following query, where T1.REVENUE is a DECIMAL(10,2) column and T2.REVENUE is a DECIMAL(8,4) column, the decimal result is promoted to DECIMAL(12,4):

```
select t1.revenue union select t2.revenue;
```

The scale is 4 because that is the maximum scale of the two columns. The precision is 12 because T1.REVENUE requires 8 digits to the left of the decimal point ( $12 - 4 = 8$ ). This type promotion ensures that all values from both sides of the UNION fit in the result. For 64-bit values, the maximum result precision is 19 and the maximum result scale is 18. For 128-bit values, the maximum result precision is 38 and the maximum result scale is 37.

If the resulting data type exceeds Amazon Redshift precision and scale limits, the query returns an error.

- For set operations, two rows are treated as identical if, for each corresponding pair of columns, the two data values are either *equal* or *both NULL*. For example, if tables T1 and T2 both contain one column and one row, and that row is NULL in both tables, an INTERSECT operation over those tables returns that row.

## Example UNION queries

In the following UNION query, rows in the SALES table are merged with rows in the LISTING table. Three compatible columns are selected from each table; in this case, the corresponding columns have the same names and data types.

The final result set is ordered by the first column in the LISTING table and limited to the 5 rows with the highest LISTID value.

```
select listid, sellerid, eventid from listing
union select listid, sellerid, eventid from sales
order by listid, sellerid, eventid desc limit 5;

listid | sellerid | eventid
-----+-----+-----
1 | 36861 | 7872
2 | 16002 | 4806
3 | 21461 | 4256
4 | 8117 | 4337
5 | 1616 | 8647
(5 rows)
```

The following example shows how you can add a literal value to the output of a UNION query so you can see which query expression produced each row in the result set. The query identifies rows from the first query expression as "B" (for buyers) and rows from the second query expression as "S" (for sellers).

The query identifies buyers and sellers for ticket transactions that cost \$10,000 or more. The only difference between the two query expressions on either side of the UNION operator is the joining column for the SALES table.

```
select listid, lastname, firstname, username,
pricepaid as price, 'S' as buyorsell
```

```

from sales, users
where sales.sellerid=users.userid
and pricepaid >=10000
union
select listid, lastname, firstname, username, pricepaid,
'B' as buyorsell
from sales, users
where sales.buyerid=users.userid
and pricepaid >=10000
order by 1, 2, 3, 4, 5;

listid | lastname | firstname | username | price | buyorsell
-----+-----+-----+-----+-----+
209658 | Lamb | Colette | V0R15LYI | 10000.00 | B
209658 | West | Kato | ELU81XAA | 10000.00 | S
212395 | Greer | Harlan | GX071KOC | 12624.00 | S
212395 | Perry | Cora | YWR73YNZ | 12624.00 | B
215156 | Banks | Patrick | ZNQ69CLT | 10000.00 | S
215156 | Hayden | Malachi | BBG56AKU | 10000.00 | B
(6 rows)

```

The following example uses a UNION ALL operator because duplicate rows, if found, need to be retained in the result. For a specific series of event IDs, the query returns 0 or more rows for each sale associated with each event, and 0 or 1 row for each listing of that event. Event IDs are unique to each row in the LISTING and EVENT tables, but there might be multiple sales for the same combination of event and listing IDs in the SALES table.

The third column in the result set identifies the source of the row. If it comes from the SALES table, it is marked "Yes" in the SALESROW column. (SALESROW is an alias for SALES.LISTID.) If the row comes from the LISTING table, it is marked "No" in the SALESROW column.

In this case, the result set consists of three sales rows for listing 500, event 7787. In other words, three different transactions took place for this listing and event combination. The other two listings, 501 and 502, did not produce any sales, so the only row that the query produces for these list IDs comes from the LISTING table (SALESROW = 'No').

```

select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union all
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
-----+-----+-----
7787 | 500 | No
7787 | 500 | Yes
7787 | 500 | Yes
7787 | 500 | Yes
6473 | 501 | No
5108 | 502 | No
(6 rows)

```

If you run the same query without the ALL keyword, the result retains only one of the sales transactions.

```

select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union
select eventid, listid, 'No'

```

```
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
-----+-----+-----
7787 | 500 | No
7787 | 500 | Yes
6473 | 501 | No
5108 | 502 | No
(4 rows)
```

## Example UNION ALL query

The following example uses a UNION ALL operator because duplicate rows, if found, need to be retained in the result. For a specific series of event IDs, the query returns 0 or more rows for each sale associated with each event, and 0 or 1 row for each listing of that event. Event IDs are unique to each row in the LISTING and EVENT tables, but there might be multiple sales for the same combination of event and listing IDs in the SALES table.

The third column in the result set identifies the source of the row. If it comes from the SALES table, it is marked "Yes" in the SALESROW column. (SALESROW is an alias for SALES.LISTID.) If the row comes from the LISTING table, it is marked "No" in the SALESROW column.

In this case, the result set consists of three sales rows for listing 500, event 7787. In other words, three different transactions took place for this listing and event combination. The other two listings, 501 and 502, did not produce any sales, so the only row that the query produces for these list IDs comes from the LISTING table (SALESROW = 'No').

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union all
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
-----+-----+-----
7787 | 500 | No
7787 | 500 | Yes
7787 | 500 | Yes
7787 | 500 | Yes
6473 | 501 | No
5108 | 502 | No
(6 rows)
```

If you run the same query without the ALL keyword, the result retains only one of the sales transactions.

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
-----+-----+-----
```

|          |     |     |
|----------|-----|-----|
| 7787     | 500 | No  |
| 7787     | 500 | Yes |
| 6473     | 501 | No  |
| 5108     | 502 | No  |
| (4 rows) |     |     |

## Example INTERSECT queries

Compare the following example with the first UNION example. The only difference between the two examples is the set operator that is used, but the results are very different. Only one of the rows is the same:

|        |       |      |
|--------|-------|------|
| 235494 | 23875 | 8771 |
|--------|-------|------|

This is the only row in the limited result of 5 rows that was found in both tables.

```
select listid, sellerid, eventid from listing
intersect
select listid, sellerid, eventid from sales
order by listid desc, sellerid, eventid
limit 5;

listid | sellerid | eventid
-----+-----+-----
235494 | 23875 | 8771
235482 | 1067 | 2667
235479 | 1589 | 7303
235476 | 15550 | 793
235475 | 22306 | 7848
(5 rows)
```

The following query finds events (for which tickets were sold) that occurred at venues in both New York City and Los Angeles in March. The difference between the two query expressions is the constraint on the VENUECITY column.

```
select distinct eventname from event, sales, venue
where event.eventid=sales.eventid and event.venueid=venue.venueid
and date_part(month,starttime)=3 and venuecity='Los Angeles'
intersect
select distinct eventname from event, sales, venue
where event.eventid=sales.eventid and event.venueid=venue.venueid
and date_part(month,starttime)=3 and venuecity='New York City'
order by eventname asc;

eventname

A Streetcar Named Desire
Dirty Dancing
Electra
Running with Annalise
Hairspray
Mary Poppins
November
Oliver!
Return To Forever
Rhinoceros
South Pacific
The 39 Steps
The Bacchae
The Caucasian Chalk Circle
The Country Girl
```

```
Wicked
Woyzeck
(16 rows)
```

### Example EXCEPT query

The CATEGORY table in the TICKIT database contains the following 11 rows:

| catid | catgroup | catname   | catdesc                                    |
|-------|----------|-----------|--------------------------------------------|
| 1     | Sports   | MLB       | Major League Baseball                      |
| 2     | Sports   | NHL       | National Hockey League                     |
| 3     | Sports   | NFL       | National Football League                   |
| 4     | Sports   | NBA       | National Basketball Association            |
| 5     | Sports   | MLS       | Major League Soccer                        |
| 6     | Shows    | Musicals  | Musical theatre                            |
| 7     | Shows    | Plays     | All non-musical theatre                    |
| 8     | Shows    | Opera     | All opera and light opera                  |
| 9     | Concerts | Pop       | All rock and pop music concerts            |
| 10    | Concerts | Jazz      | All jazz singers and bands                 |
| 11    | Concerts | Classical | All symphony, concerto, and choir concerts |

(11 rows)

Assume that a CATEGORY\_STAGE table (a staging table) contains one additional row:

| catid | catgroup | catname   | catdesc                                    |
|-------|----------|-----------|--------------------------------------------|
| 1     | Sports   | MLB       | Major League Baseball                      |
| 2     | Sports   | NHL       | National Hockey League                     |
| 3     | Sports   | NFL       | National Football League                   |
| 4     | Sports   | NBA       | National Basketball Association            |
| 5     | Sports   | MLS       | Major League Soccer                        |
| 6     | Shows    | Musicals  | Musical theatre                            |
| 7     | Shows    | Plays     | All non-musical theatre                    |
| 8     | Shows    | Opera     | All opera and light opera                  |
| 9     | Concerts | Pop       | All rock and pop music concerts            |
| 10    | Concerts | Jazz      | All jazz singers and bands                 |
| 11    | Concerts | Classical | All symphony, concerto, and choir concerts |
| 12    | Concerts | Comedy    | All stand up comedy performances           |

(12 rows)

Return the difference between the two tables. In other words, return rows that are in the CATEGORY\_STAGE table but not in the CATEGORY table:

```
select * from category_stage
except
select * from category;

catid | catgroup | catname | catdesc
-----+-----+-----+-----+
12 | Concerts | Comedy | All stand up comedy performances
(1 row)
```

The following equivalent query uses the synonym MINUS.

```
select * from category_stage
minus
select * from category;

catid | catgroup | catname | catdesc
```

```
+-----+-----+-----+
12 | Concerts | Comedy | All stand up comedy performances
(1 row)
```

If you reverse the order of the SELECT expressions, the query returns no rows.

## ORDER BY clause

### Topics

- [Syntax \(p. 969\)](#)
- [Parameters \(p. 969\)](#)
- [Usage notes \(p. 970\)](#)
- [Examples with ORDER BY \(p. 970\)](#)

The ORDER BY clause sorts the result set of a query.

### Syntax

```
[ORDER BY expression [ASC | DESC]]
[NULLS FIRST | NULLS LAST]
[LIMIT { count | ALL }]
[OFFSET start]
```

### Parameters

#### *expression*

Expression that defines the sort order of the query result set, typically by specifying one or more columns in the select list. Results are returned based on binary UTF-8 ordering. You can also specify the following:

- Columns that aren't in the select list
- Expressions formed from one or more columns that exist in the tables referenced by the query
- Ordinal numbers that represent the position of select list entries (or the position of columns in the table if no select list exists)
- Aliases that define select list entries

When the ORDER BY clause contains multiple expressions, the result set is sorted according to the first expression, then the second expression is applied to rows that have matching values from the first expression, and so on.

#### ASC | DESC

Option that defines the sort order for the expression, as follows:

- ASC: ascending (for example, low to high for numeric values and 'A' to 'Z' for character strings). If no option is specified, data is sorted in ascending order by default.
- DESC: descending (high to low for numeric values; 'Z' to 'A' for strings).

#### NULLS FIRST | NULLS LAST

Option that specifies whether NULL values should be ordered first, before non-null values, or last, after non-null values. By default, NULL values are sorted and ranked last in ASC ordering, and sorted and ranked first in DESC ordering.

#### **LIMIT *number* | ALL**

Option that controls the number of sorted rows that the query returns. The LIMIT number must be a positive integer; the maximum value is 2147483647.

LIMIT 0 returns no rows. You can use this syntax for testing purposes: to check that a query runs (without displaying any rows) or to return a column list from a table. An ORDER BY clause is redundant if you are using LIMIT 0 to return a column list. The default is LIMIT ALL.

#### **OFFSET *start***

Option that specifies to skip the number of rows before *start* before beginning to return rows. The OFFSET number must be a positive integer; the maximum value is 2147483647. When used with the LIMIT option, OFFSET rows are skipped before starting to count the LIMIT rows that are returned. If the LIMIT option isn't used, the number of rows in the result set is reduced by the number of rows that are skipped. The rows skipped by an OFFSET clause still have to be scanned, so it might be inefficient to use a large OFFSET value.

### Usage notes

Note the following expected behavior with ORDER BY clauses:

- NULL values are considered "higher" than all other values. With the default ascending sort order, NULL values sort at the end. To change this behavior, use the NULLS FIRST option.
- When a query doesn't contain an ORDER BY clause, the system returns result sets with no predictable ordering of the rows. The same query run twice might return the result set in a different order.
- The LIMIT and OFFSET options can be used without an ORDER BY clause; however, to return a consistent set of rows, use these options in conjunction with ORDER BY.
- In any parallel system like Amazon Redshift, when ORDER BY doesn't produce a unique ordering, the order of the rows is nondeterministic. That is, if the ORDER BY expression produces duplicate values, the return order of those rows might vary from other systems or from one run of Amazon Redshift to the next.
- Amazon Redshift doesn't support string literals in ORDER BY clauses.

### Examples with ORDER BY

Return all 11 rows from the CATEGORY table, ordered by the second column, CATGROUP. For results that have the same CATGROUP value, order the CATDESC column values by the length of the character string. Then order by columns CATID and CATNAME.

```
select * from category order by 2, length(catdesc), 1, 3;

catid | catgroup | catname | catdesc
-----+-----+-----+
10 | Concerts | Jazz | All jazz singers and bands
9 | Concerts | Pop | All rock and pop music concerts
11 | Concerts | Classical | All symphony, concerto, and choir conce
6 | Shows | Musicals | Musical theatre
7 | Shows | Plays | All non-musical theatre
8 | Shows | Opera | All opera and light opera
5 | Sports | MLS | Major League Soccer
1 | Sports | MLB | Major League Baseball
2 | Sports | NHL | National Hockey League
3 | Sports | NFL | National Football League
4 | Sports | NBA | National Basketball Association
(11 rows)
```

Return selected columns from the SALES table, ordered by the highest QTYSOLD values. Limit the result to the top 10 rows:

```
select salesid, qtysold, pricepaid, commission, saletime from sales
order by qtysold, pricepaid, commission, salesid, saletime desc
limit 10;

salesid | qtysold | pricepaid | commission | saletime
-----+-----+-----+-----+
15401 | 8 | 272.00 | 40.80 | 2008-03-18 06:54:56
61683 | 8 | 296.00 | 44.40 | 2008-11-26 04:00:23
90528 | 8 | 328.00 | 49.20 | 2008-06-11 02:38:09
74549 | 8 | 336.00 | 50.40 | 2008-01-19 12:01:21
130232 | 8 | 352.00 | 52.80 | 2008-05-02 05:52:31
55243 | 8 | 384.00 | 57.60 | 2008-07-12 02:19:53
16004 | 8 | 440.00 | 66.00 | 2008-11-04 07:22:31
489 | 8 | 496.00 | 74.40 | 2008-08-03 05:48:55
4197 | 8 | 512.00 | 76.80 | 2008-03-23 11:35:33
16929 | 8 | 568.00 | 85.20 | 2008-12-19 02:59:33
(10 rows)
```

Return a column list and no rows by using LIMIT 0 syntax:

```
select * from venue limit 0;
venueid | venuename | venuecity | venuestate | venueseats
-----+-----+-----+-----+
(0 rows)
```

## CONNECT BY clause

The CONNECT BY clause specifies the relationship between rows in a hierarchy. You can use CONNECT BY to select rows in a hierarchical order by joining the table to itself and processing the hierarchical data. For example, you can use it to recursively loop through an organization chart and list data.

Hierarchical queries process in the following order:

1. If the FROM clause has a join, it is processed first.
2. The CONNECT BY clause is evaluated.
3. The WHERE clause is evaluated.

## Syntax

```
[START WITH start_with_conditions]
CONNECT BY connect_by_conditions
```

### Note

While START and CONNECT are not reserved words, use delimited identifiers (double quotation marks) or AS if you're using START and CONNECT as table aliases in your query to avoid failure at runtime.

```
SELECT COUNT(*)
FROM Employee "start"
CONNECT BY PRIOR id = manager_id
START WITH name = 'John'
```

```
SELECT COUNT(*)
```

```
FROM Employee AS start
CONNECT BY PRIOR id = manager_id
START WITH name = 'John'
```

## Parameters

*start\_with\_conditions*

Conditions that specify the root row(s) of the hierarchy

*connect\_by\_conditions*

Conditions that specify the relationship between parent rows and child rows of the hierarchy. At least one condition must be qualified with the PRIOR unary operator used to refer to the parent row.

```
PRIOR column = expression
-- or
expression > PRIOR column
```

## Operators

You can use the following operators in a CONNECT BY query.

*LEVEL*

Pseudocolumn that returns the current row level in the hierarchy. Returns 1 for the root row, 2 for the child of the root row, and so on.

*PRIOR*

Unary operator that evaluates the expression for the parent row of the current row in the hierarchy.

## Examples

The following example is a CONNECT BY query that returns the number of employees that report directly or indirectly to John, no deeper than 4 levels.

```
select id, name, manager_id
from employee
where level < 4
start with name = 'John'
connect by prior id = manager_id;
```

Following is the result of the query.

| id  | name    | manager_id |
|-----|---------|------------|
| 101 | John    | 100        |
| 102 | Jorge   | 101        |
| 103 | Kwaku   | 101        |
| 110 | Liu     | 101        |
| 201 | Sofía   | 102        |
| 106 | Mateo   | 102        |
| 110 | Nikki   | 103        |
| 104 | Paulo   | 103        |
| 105 | Richard | 103        |
| 120 | Saanvi  | 104        |
| 200 | Shirley | 104        |
| 205 | Zhang   | 104        |

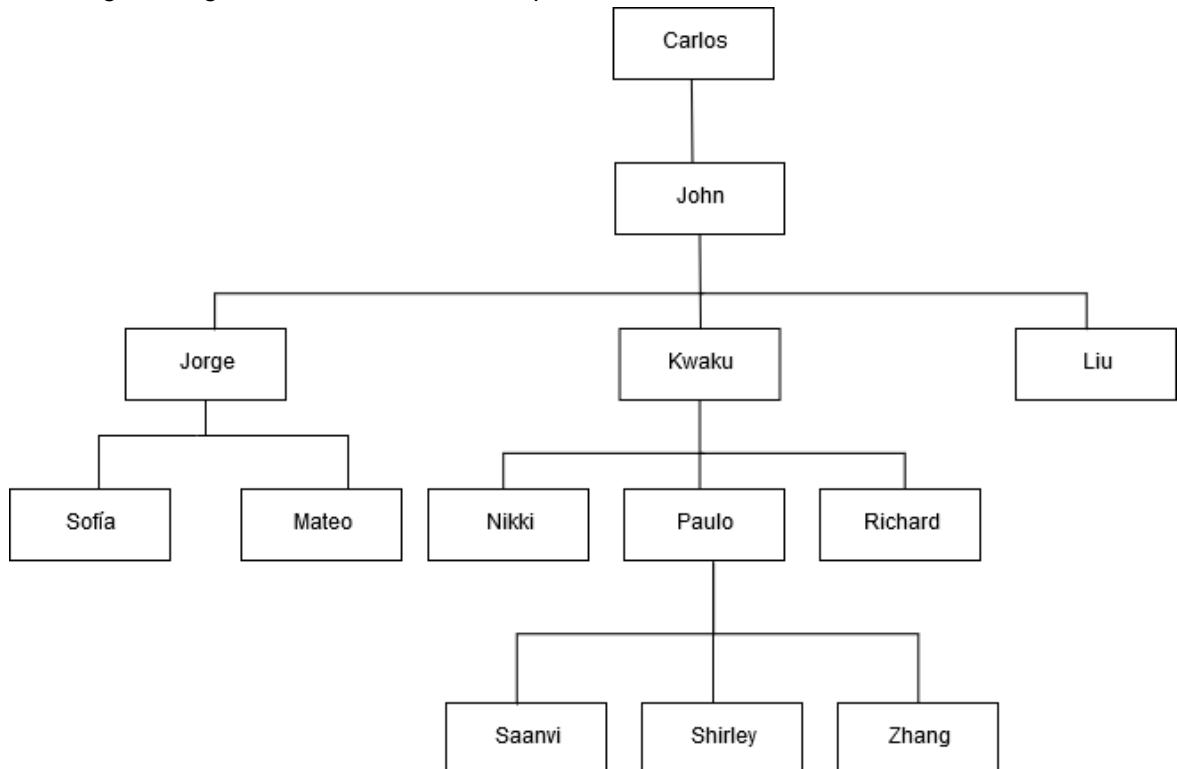
Table definition for this example:

```
create table employee (
 id int,
 name varchar (20),
 manager_id int
);
```

Following are the rows inserted into the table.

```
insert into employee(id, name, manager_id) values
(100, 'Carlos', null),
(101, 'John', 100),
(102, 'Jorge', 101),
(103, 'Kwaku', 101),
(110, 'Liu', 101),
(106, 'Mateo', 102),
(110, 'Nikki', 103),
(104, 'Paulo', 103),
(105, 'Richard', 103),
(120, 'Saanvi', 104),
(200, 'Shirley', 104),
(201, 'Sofia', 102),
(205, 'Zhang', 104);
```

Following is an organization chart for John's department.



## Subquery examples

The following examples show different ways in which subqueries fit into SELECT queries. See [JOIN examples \(p. 946\)](#) for another example of the use of subqueries.

## SELECT list subquery

The following example contains a subquery in the SELECT list. This subquery is *scalar*: it returns only one column and one value, which is repeated in the result for each row that is returned from the outer query. The query compares the Q1SALES value that the subquery computes with sales values for two other quarters (2 and 3) in 2008, as defined by the outer query.

```
select qtr, sum(pricepaid) as qtrsales,
(select sum(pricepaid)
from sales join date on sales.dateid=date.dateid
where qtr='1' and year=2008) as q1sales
from sales join date on sales.dateid=date.dateid
where qtr in('2','3') and year=2008
group by qtr
order by qtr;

qtr | qtrsales | q1sales
-----+-----+-----
2 | 30560050.00 | 24742065.00
3 | 31170237.00 | 24742065.00
(2 rows)
```

## WHERE clause subquery

The following example contains a table subquery in the WHERE clause. This subquery produces multiple rows. In this case, the rows contain only one column, but table subqueries can contain multiple columns and rows, just like any other table.

The query finds the top 10 sellers in terms of maximum tickets sold. The top 10 list is restricted by the subquery, which removes users who live in cities where there are ticket venues. This query can be written in different ways; for example, the subquery could be rewritten as a join within the main query.

```
select firstname, lastname, city, max(qtysold) as maxsold
from users join sales on users.userid=sales.sellerid
where users.city not in(select venuecity from venue)
group by firstname, lastname, city
order by maxsold desc, city desc
limit 10;

firstname | lastname | city | maxsold
-----+-----+-----+-----+
Noah | Guerrero | Worcester | 8
Isadora | Moss | Winooski | 8
Kieran | Harrison | Westminster | 8
Heidi | Davis | Warwick | 8
Sara | Anthony | Waco | 8
Bree | Buck | Valdez | 8
Evangeline | Sampson | Trenton | 8
Kendall | Keith | Stillwater | 8
Bertha | Bishop | Stevens Point | 8
Patricia | Anderson | South Portland | 8
(10 rows)
```

## WITH clause subqueries

See [WITH clause \(p. 929\)](#).

## Correlated subqueries

The following example contains a *correlated subquery* in the WHERE clause; this kind of subquery contains one or more correlations between its columns and the columns produced by the outer query. In

this case, the correlation is where `s.listid=l.listid`. For each row that the outer query produces, the subquery is run to qualify or disqualify the row.

```
select salesid, listid, sum(pricepaid) from sales s
where qtysold=
(select max(numtickets) from listing l
where s.listid=l.listid)
group by 1,2
order by 1,2
limit 5;

salesid | listid | sum
-----+-----+-----
27 | 28 | 111.00
81 | 103 | 181.00
142 | 149 | 240.00
146 | 152 | 231.00
194 | 210 | 144.00
(5 rows)
```

## Correlated subquery patterns that are not supported

The query planner uses a query rewrite method called subquery decorrelation to optimize several patterns of correlated subqueries for execution in an MPP environment. A few types of correlated subqueries follow patterns that Amazon Redshift can't decorrelate and doesn't support. Queries that contain the following correlation references return errors:

- Correlation references that skip a query block, also known as "skip-level correlation references." For example, in the following query, the block containing the correlation reference and the skipped block are connected by a NOT EXISTS predicate:

```
select event.eventname from event
where not exists
(select * from listing
where not exists
(select * from sales where event.eventid=sales.eventid));
```

The skipped block in this case is the subquery against the LISTING table. The correlation reference correlates the EVENT and SALES tables.

- Correlation references from a subquery that is part of an ON clause in an outer join:

```
select * from category
left join event
on category.catid=event.catid and eventid =
(select max(eventid) from sales where sales.eventid=event.eventid);
```

The ON clause contains a correlation reference from SALES in the subquery to EVENT in the outer query.

- Null-sensitive correlation references to an Amazon Redshift system table. For example:

```
select attrelid
from stv_locks sl, pg_attribute
where sl.table_id=pg_attribute.attrelid and 1 not in
(select 1 from pg_opclass where sl.lock_owner = opcowner);
```

- Correlation references from within a subquery that contains a window function.

```
select listid, qtysold
```

```
from sales s
where qtypsold not in
(select sum(numtickets) over() from listing l where s.listid=l.listid);
```

- References in a GROUP BY column to the results of a correlated subquery. For example:

```
select listing.listid,
(select count(sales.listid) from sales where sales.listid=listing.listid) as list
from listing
group by list, listing.listid;
```

- Correlation references from a subquery with an aggregate function and a GROUP BY clause, connected to the outer query by an IN predicate. (This restriction doesn't apply to MIN and MAX aggregate functions.) For example:

```
select * from listing where listid in
(select sum(qtypsold)
from sales
where numtickets>4
group by salesid);
```

## SELECT INTO

Selects rows defined by any query and inserts them into a new table. You can specify whether to create a temporary or a persistent table.

### Syntax

```
[WITH with_subquery [, ...]]
SELECT
[TOP number] [ALL | DISTINCT]
* | expression [AS output_name] [, ...]
INTO [TEMPORARY | TEMP] [TABLE] new_table
[FROM table_reference [, ...]]
[WHERE condition]
[GROUP BY expression [, ...]]
[HAVING condition [, ...]]
[{ UNION | INTERSECT | { EXCEPT | MINUS } } [ALL] query]
[ORDER BY expression
[ASC | DESC]
[LIMIT { number | ALL }]
[OFFSET start]
```

For details about the parameters of this command, see [SELECT \(p. 928\)](#).

### Examples

Select all of the rows from the EVENT table and create a NEWEVENT table:

```
select * into newevent from event;
```

Select the result of an aggregate query into a temporary table called PROFITS:

```
select username, lastname, sum(pricepaid-commission) as profit
into temp table profits
from sales, users
```

```
where sales.sellerid=users.userid
group by 1, 2
order by 3 desc;
```

## SET

Sets the value of a server configuration parameter. Use the SET command to override a setting for the duration of the current session or transaction only.

Use the [RESET \(p. 915\)](#) command to return a parameter to its default value.

You can change the server configuration parameters in several ways. For more information, see [Modifying the server configuration \(p. 1685\)](#).

## Syntax

```
SET { [SESSION | LOCAL]
{ SEED | parameter_name } { TO | = }
{ value | 'value' | DEFAULT } |
SEED TO value }
```

The following statement sets the value of a session context variable.

```
SET { [SESSION | LOCAL]
variable_name { TO | = }
{ value | 'value' }
```

## Parameters

### SESSION

Specifies that the setting is valid for the current session. Default value.

#### *variable\_name*

Specifies the name of the context variable set for the session.

The naming convention is a two-part name separated by a dot, for example *identifier.identifier*. Only one dot separator is allowed. Use an *identifier* that follows the standard identifier rules for Amazon Redshift. For more information, see [Names and identifiers \(p. 571\)](#). Delimited identifiers aren't allowed.

### LOCAL

Specifies that the setting is valid for the current transaction.

#### SEED TO *value*

Sets an internal seed to be used by the RANDOM function for random number generation.

SET SEED takes a numeric *value* between 0 and 1, and multiplies this number by  $(2^{31}-1)$  for use with the [RANDOM function \(p. 1170\)](#) function. If you use SET SEED before making multiple RANDOM calls, RANDOM generates numbers in a predictable sequence.

#### *parameter\_name*

Name of the parameter to set. See [Modifying the server configuration \(p. 1685\)](#) for information about parameters.

*value*

New parameter value. Use single quotation marks to set the value to a specific string. If using SET SEED, this parameter contains the SEED value.

**DEFAULT**

Sets the parameter to the default value.

## Examples

### Changing a parameter for the current session

The following example sets the datestyle:

```
set datestyle to 'SQL,DMY';
```

### Setting a query group for workload management

If query groups are listed in a queue definition as part of the cluster's WLM configuration, you can set the QUERY\_GROUP parameter to a listed query group name. Subsequent queries are assigned to the associated query queue. The QUERY\_GROUP setting remains in effect for the duration of the session or until a RESET QUERY\_GROUP command is encountered.

This example runs two queries as part of the query group 'priority', then resets the query group.

```
set query_group to 'priority';
select tbl, count(*)from stv_blocklist;
select query, elapsed, substring from svl_qlog order by query desc limit 5;
reset query_group;
```

See [Implementing workload management \(p. 491\)](#)

### Setting a label for a group of queries

The QUERY\_GROUP parameter defines a label for one or more queries that are run in the same session after a SET command. In turn, this label is logged when queries are run and can be used to constrain results returned from the STL\_QUERY and STV\_INFLIGHT system tables and the SVL\_QLOG view.

```
show query_group;
query_group

unset
(1 row)

set query_group to '6 p.m.';

show query_group;
query_group

6 p.m.
(1 row)

select * from sales where salesid=500;
salesid | listid | sellerid | buyerid | eventid | dateid | ...
-----+-----+-----+-----+-----+-----+
500 | 504 | 3858 | 2123 | 5871 | 2052 | ...
```

```
(1 row)

reset query_group;

select query, trim(label) querygroup, pid, trim(querytxt) sql
from stl_query
where label ='6 p.m.';
query | querygroup | pid | sql
-----+-----+-----+
57 | 6 p.m. | 30711 | select * from sales where salesid=500;
(1 row)
```

Query group labels are a useful mechanism for isolating individual queries or groups of queries that are run as part of scripts. You don't need to identify and track queries by their IDs; you can track them by their labels.

#### **Setting a seed value for random number generation**

The following example uses the SEED option with SET to cause the RANDOM function to generate numbers in a predictable sequence.

First, return three RANDOM integers without setting the SEED value first:

```
select cast (random() * 100 as int);
int4

6
(1 row)

select cast (random() * 100 as int);
int4

68
(1 row)

select cast (random() * 100 as int);
int4

56
(1 row)
```

Now, set the SEED value to .25, and return three more RANDOM numbers:

```
set seed to .25;

select cast (random() * 100 as int);
int4

21
(1 row)

select cast (random() * 100 as int);
int4

79
(1 row)

select cast (random() * 100 as int);
int4

12
(1 row)
```

Finally, reset the SEED value to .25, and verify that RANDOM returns the same results as the previous three calls:

```
set seed to .25;

select cast (random() * 100 as int);
int4

21
(1 row)

select cast (random() * 100 as int);
int4

79
(1 row)

select cast (random() * 100 as int);
int4

12
(1 row)
```

The following example sets a customized context variable.

```
SET app_context.user_id TO 123;
SET app_context.user_id TO 'sample_variable_value';
```

## SET SESSION AUTHORIZATION

Sets the user name for the current session.

You can use the SET SESSION AUTHORIZATION command, for example, to test database access by temporarily running a session or transaction as an unprivileged user. You must be a database superuser to run this command.

### Syntax

```
SET [LOCAL] SESSION AUTHORIZATION { user_name | DEFAULT }
```

### Parameters

#### LOCAL

Specifies that the setting is valid for the current transaction. Omitting this parameter specifies that the setting is valid for the current session.

#### *user\_name*

Name of the user to set. The user name may be written as an identifier or a string literal.

#### DEFAULT

Sets the session user name to the default value.

### Examples

The following example sets the user name for the current session to dwuser:

```
SET SESSION AUTHORIZATION 'dwuser';
```

The following example sets the user name for the current transaction to dwuser:

```
SET LOCAL SESSION AUTHORIZATION 'dwuser';
```

This example sets the user name for the current session to the default user name:

```
SET SESSION AUTHORIZATION DEFAULT;
```

## SET SESSION CHARACTERISTICS

This command is deprecated.

## SHOW

Displays the current value of a server configuration parameter. This value may be specific to the current session if a SET command is in effect. For a list of configuration parameters, see [Configuration reference \(p. 1685\)](#).

### Syntax

```
SHOW { parameter_name | ALL }
```

The following statement displays the current value of a session context variable. If the variable doesn't exist, Amazon Redshift throws an error.

```
SHOW variable_name
```

### Parameters

*parameter\_name*

Displays the current value of the specified parameter.

ALL

Displays the current values of all of the parameters.

*variable\_name*

Displays the current value of the specified variable.

## Examples

The following example displays the value for the query\_group parameter:

```
show query_group;
query_group
unset
```

(1 row)

The following example displays a list of all parameters and their values:

```
show all;
name | setting

datestyle | ISO, MDY
extra_float_digits | 0
query_group | unset
search_path | $user,public
statement_timeout | 0
```

The following example displays the current value of the specified variable.

```
SHOW app_context.user_id;
```

## SHOW EXTERNAL TABLE

Shows the definition of an external table, including table attributes and column attributes. You can use the output of the SHOW EXTERNAL TABLE statement to recreate the table.

For more information about external table creation, see [CREATE EXTERNAL TABLE \(p. 768\)](#).

### Syntax

```
SHOW EXTERNAL TABLE external_schema.table_name [PARTITION]
```

### Parameters

*external\_schema*

The name of the associated external schema.

*table\_name*

The name of the table to show.

PARTITION

Displays ALTER TABLE statements to add partitions to the table definition.

### Examples

The following examples are based on an external table defined as follows:

```
CREATE EXTERNAL TABLE my_schema.alldatatype_parquet_test_partitioned (
 csmallint smallint,
 cint int,
 cbigint bigint,
 cfloat float4,
 cdouble float8,
 cchar char(10),
 cvarchar varchar(255),
 cdecimal_small decimal(18,9),
```

```

 cdecimal_big decimal(30,15),
 ctimestamp TIMESTAMP,
 cboolean boolean,
 cstring varchar(16383)
)
PARTITIONED BY (cdate date, ctime TIMESTAMP)
STORED AS PARQUET
LOCATION 's3://mybucket-test-copy/alldatatypes_parquet_partitioned';

```

Following is an example of the SHOW EXTERNAL TABLE command and output for the table my\_schema.alldatatypes\_parquet\_test\_partitioned.

```
SHOW EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned;
```

```

"CREATE EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned (
 csmallint smallint,
 cint int,
 cbigint bigint,
 cfloat float4,
 cdouble float8,
 cchar char(10),
 cvarchar varchar(255),
 cdecimal_small decimal(18,9),
 cdecimal_big decimal(30,15),
 ctimestamp timestamp,
 cboolean boolean,
 cstring varchar(16383)
)
PARTITIONED BY (cdate date, ctime timestamp)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://mybucket-test-copy/alldatatypes_parquet_partitioned';"

```

Following is an example of the SHOW EXTERNAL TABLE command and output when using the PARTITION parameter. The output contains ALTER TABLE statements to add partitions to the table definition.

```
SHOW EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned PARTITION;
```

```

"CREATE EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned (
 csmallint smallint,
 cint int,
 cbigint bigint,
 cfloat float4,
 cdouble float8,
 cchar char(10),
 cvarchar varchar(255),
 cdecimal_small decimal(18,9),
 cdecimal_big decimal(30,15),
 ctimestamp timestamp,
 cboolean boolean,
 cstring varchar(16383)
)
PARTITIONED BY (cdate date)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://mybucket-test-copy/alldatatypes_parquet_partitioned';

```

```
ALTER TABLE my_schema.alldatatypes_parquet_test_partitioned ADD IF NOT EXISTS PARTITION
 (cdate='2021-01-01') LOCATION 's3://mybucket-test-copy/alldatatypes_parquet_partitioned2/
 cdate=2021-01-01';
ALTER TABLE my_schema.alldatatypes_parquet_test_partitioned ADD IF NOT EXISTS PARTITION
 (cdate='2021-01-02') LOCATION 's3://mybucket-test-copy/alldatatypes_parquet_partitioned2/
 cdate=2021-01-02';"
```

## SHOW DATABASES

Displays databases from a specified account ID.

### Syntax

```
SHOW DATABASES FROM
DATA CATALOG [ACCOUNT '<id1>', '<id2>', ...]
[LIKE '<expression>']
```

### Parameters

**ACCOUNT '<id1>', '<id2>', ...**

The AWS Glue Data Catalog accounts from which to list databases. Omitting this parameter indicates that Amazon Redshift should show the databases from the account that owns the cluster.

**LIKE '<expression>'**

Filters the list of databases to those that match the expression that you specify. This parameter supports patterns that use the wildcard characters % (percent) and \_ (underscore).

### Examples

The following example displays all of the Data Catalog databases from the account ID 123456789012.

```
SHOW DATABASES FROM DATA CATALOG ACCOUNT '123456789012'
catalog_id | database_name | database_arn
type | | target_database
 | location | parameters
-----+-----+-----+
+-----+
+-----+
+-----+
123456789012 | database1 | arn:aws:glue:us-east-1:123456789012:database/database1 |
Data Catalog | | |
 | |
123456789012 | database2 | arn:aws:glue:us-east-1:123456789012:database/database2 |
Data Catalog | arn:aws:redshift:us- | |
east-1:123456789012:datashare:035c45ea-61ce-86f0-8b75-19ac6102c3b7/database2 |
```

## SHOW MODEL

Shows useful information about a machine learning model, including its status, the parameters used to create it and the prediction function with its input argument types. You can use the information from the SHOW MODEL to recreate the model. If base tables have changed, running CREATE MODEL with the same SQL statement results in a different model. The information returned by the SHOW MODEL is different for the model owner and a user with the EXECUTE privilege. SHOW MODEL shows different outputs when a model is trained from Amazon Redshift or when the model is a BYOM model.

## Syntax

```
SHOW MODEL (ALL | model_name)
```

## Parameters

ALL

Returns all the models that the user can use and their schemas.

*model\_name*

The name of the model. The model name in a schema must be unique.

## Usage notes

The SHOW MODEL command returns the following:

- The model name.
- The schema where the model was created.
- The owner of the model.
- The model creation time.
- The status of the model, such as READY, TRAINING, or FAILED.
- The reason message for a failed model.
- The validation error if model has finished training.
- The estimated cost needed to derive the model for a non-BYOM approach. Only the owner of the model can view this information.
- A list of user-specified parameters and their values, specifically the following:
  - The specified TARGET column.
  - The model type, AUTO or XGBoost.
  - The problem type, such as REGRESSION, BINARY\_CLASSIFICATION, MULTICLASS\_CLASSIFICATION. This parameter is specific to AUTO.
  - The name of the Amazon SageMaker training job or the Amazon SageMaker Autopilot job that created the model. You can use this job name to find more information about the model on Amazon SageMaker.
  - The objective, such as MSE, F1, Accuracy. This parameter is specific to AUTO.
  - The name of the created function.
  - The type of inference, local or remote.
  - The prediction function input arguments.
  - The prediction function input argument types for models that aren't bring your own model (BYOM).
  - The return type of the prediction function. This parameter is specific to BYOM.
  - The name of the Amazon SageMaker endpoint for a BYOM model with remote inference.
  - The IAM role. Only the owner of the model can see this.
  - The S3 bucket used. Only the owner of the model can see this.
  - The AWS KMS key, if one was provided. Only the owner of the model can see this.
  - The maximum time that the model can run.
- If the model type is not AUTO, then Amazon Redshift also shows the list of hyperparameters provided and their values.

You can also view some of the information provided by SHOW MODEL in other catalog tables, such as pg\_proc. Amazon Redshift returns information about the prediction function that is registered in pg\_proc catalog table. This information includes the input argument names and their types for the prediction function. Amazon Redshift returns the same information in the SHOW MODEL command.

```
SELECT * FROM pg_proc WHERE proname ILIKE '%<function_name>';
```

## Examples

The following example shows the show model output.

```
SHOW MODEL ALL;

Schema Name | Model Name
-----+-----
public | customer_churn
```

The owner of the customer\_churn can see the following output. A user with only the EXECUTE privilege can't see the IAM role, the Amazon S3 bucket, and the estimated cost of the mode.

```
SHOW MODEL customer_churn;

Key | Value
-----+-----
Model Name | customer_churn
Schema Name | public
Owner | 'owner'
Creation Time | Sat, 15.01.2000 14:45:20
Model State | READY
validation:F1 | 0.855
Estimated Cost | 5.7

TRAINING DATA:
Table | customer_data
Target Column | CHURN

PARAMETERS:
Model Type | auto
Problem Type | binary_classification
Objective | f1
Function Name | predict_churn
Function Parameters | age zip average_daily_spend average_daily_cases
Function Parameter Types | int int float float
IAM Role | 'iam_role'
KMS Key | 'kms_key'
Max Runtime | 36000
```

## SHOW DATASHARES

Displays the inbound and outbound shares in a cluster either from the same account or across accounts. If you don't specify a datashare name, then Amazon Redshift displays all datashares in all databases in the cluster. Users who have the ALTER and SHARE privileges can see the shares that they have privileges for.

## Syntax

```
SHOW DATASHARES [LIKE 'namepattern']
```

## Parameters

**LIKE**

An optional clause that compares the specified name pattern to the description of the datashare. When this clause is used, Amazon Redshift displays only the datashares with names that match the specified name pattern.

*namepattern*

The name of the datashare requested or part of the name to be matched using wildcard characters.

## Examples

The following example displays the inbound and outbound shares in a cluster.

```
SHOW DATASHARES;
SHOW DATASHARES LIKE 'sales%';

share_name | share_owner | source_database | consumer_database | share_type |
createdate | is_publicaccessible | share_acl | producer_account |
producer_namespace
-----+-----+-----+-----+
-----+-----+-----+-----+
-----+-----+
'salesshare' | 100 | dev | | outbound
| 2020-12-09 01:22:54. | False | | 123456789012 |
13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

## SHOW PROCEDURE

Shows the definition of a given stored procedure, including its signature. You can use the output of a SHOW PROCEDURE to recreate the stored procedure.

## Syntax

```
SHOW PROCEDURE sp_name [([[[argname] [argmode] argtype [, ...]])]
```

## Parameters

*sp\_name*

The name of the procedure to show.

*[argname] [ argmode] argtype*

Input argument types to identify the stored procedure. Optionally, you can include the full argument data types, including OUT arguments. This part is optional if the name of the stored procedure is unique (that is, not overloaded).

## Examples

The following example shows the definition of the procedure test\_sp12.

```
show procedure test_sp2(int, varchar);
```

| Stored Procedure Definition                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> CREATE OR REPLACE PROCEDURE public.test_sp2(f1 integer, INOUT f2 character varying, OUT character varying) LANGUAGE plpgsql AS \$\$ DECLARE out_var alias for \$3; loop_var int; BEGIN IF f1 is null OR f2 is null THEN RAISE EXCEPTION 'input cannot be null'; END IF; CREATE TEMP TABLE etl(a int, b varchar); FOR loop_var IN 1..f1 LOOP insert into etl values (loop_var, f2); f2 := f2    '+'    f2; END LOOP; SELECT INTO out_var count(*) from etl; END; \$\$ (1 row) </pre> |

## SHOW TABLE

Shows the definition of a table, including table attributes, table constraints, column attributes, and column constraints. You can use the output of the SHOW TABLE statement to recreate the table.

For more information on table creation, see [CREATE TABLE \(p. 823\)](#).

### Syntax

|                                                |
|------------------------------------------------|
| <pre>SHOW TABLE [schema_name.]table_name</pre> |
|------------------------------------------------|

### Parameters

*schema\_name*

(Optional) The name of the related schema.

*table\_name*

The name of the table to show.

### Examples

Following is an example of the SHOW TABLE output for the table sales.

|                              |
|------------------------------|
| <pre>show table sales;</pre> |
|------------------------------|

|                                                                                                                                                                                                                                                |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> CREATE TABLE public.sales ( salesid integer NOT NULL ENCODE az64, listid integer NOT NULL ENCODE az64 distkey, sellerid integer NOT NULL ENCODE az64, buyerid integer NOT NULL ENCODE az64, eventid integer NOT NULL ENCODE az64, </pre> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```
dateid smallint NOT NULL,
qtysold smallint NOT NULL ENCODE az64,
pricepaid numeric(8,2) ENCODE az64,
commission numeric(8,2) ENCODE az64,
saletime timestamp without time zone ENCODE az64
)
DISTSTYLE KEY SORTKEY (dateid);
```

Following is an example of the SHOW TABLE output for the table category in the schema public.

```
show table public.category;
```

```
CREATE TABLE public.category (
catid smallint NOT NULL distkey,
catgroup character varying(10) ENCODE lzo,
catname character varying(10) ENCODE lzo,
catdesc character varying(50) ENCODE lzo
) DISTSTYLE KEY SORTKEY (catid);
```

The following example creates table foo with a primary key.

```
create table foo(a int PRIMARY KEY, b int);
```

The SHOW TABLE results display the create statement with all properties of the foo table.

```
show table foo;
```

```
CREATE TABLE public.foo (a integer NOT NULL ENCODE az64, b integer ENCODE az64, PRIMARY
KEY (a)) DISTSTYLE AUTO;
```

## SHOW VIEW

Shows the definition of a view, including for materialized views and late-binding views. You can use the output of the SHOW VIEW statement to recreate the view.

### Syntax

```
SHOW VIEW [schema_name.]view_name
```

### Parameters

*schema\_name*

(Optional) The name of the related schema.

*view\_name*

The name of the view to show.

### Examples

Following is the view definition for the view LA\_Venues\_v.

```
create view LA_Venues_v as select * from venue where venuecity='Los Angeles';
```

Following is an example of the SHOW VIEW command and output for the view defined preceding.

```
show view LA_Venues_v;
```

```
SELECT venue.venueid,
venue.venuename,
venue.venuecity,
venue.venuestate,
venue.venueseats
FROM venue WHERE ((venue.venuecity)::text = 'Los Angeles'::text);
```

Following is the view definition for the view public.Sports\_v in the schema public.

```
create view public.Sports_v as select * from category where catgroup='Sports';
```

Following is an example of the SHOW VIEW command and output for the view defined preceding.

```
show view public.Sports_v;
```

```
SELECT category.catid,
category.catgroup,
category.catname,
category.catdesc
FROM category WHERE ((category.catgroup)::text = 'Sports'::text);
```

## START TRANSACTION

Synonym of the BEGIN function.

See [BEGIN \(p. 669\)](#).

## TRUNCATE

Deletes all of the rows from a table without doing a table scan: this operation is a faster alternative to an unqualified DELETE operation. To run a TRUNCATE command, you must be the owner of the table or a superuser.

TRUNCATE is much more efficient than DELETE and doesn't require a VACUUM and ANALYZE. However, be aware that TRUNCATE commits the transaction in which it is run.

### Syntax

```
TRUNCATE [TABLE] table_name
```

### Parameters

TABLE

Optional keyword.

*table\_name*

A temporary or persistent table. Only the owner of the table or a superuser may truncate it.

You can truncate any table, including tables that are referenced in foreign-key constraints.

You don't need to vacuum a table after truncating it.

## Usage notes

The TRUNCATE command commits the transaction in which it is run; therefore, you can't roll back a TRUNCATE operation, and a TRUNCATE command may commit other operations when it commits itself.

## Examples

Use the TRUNCATE command to delete all of the rows from the CATEGORY table:

```
truncate category;
```

Attempt to roll back a TRUNCATE operation:

```
begin;
truncate date;
rollback;

select count(*) from date;
count

0
(1 row)
```

The DATE table remains empty after the ROLLBACK command because the TRUNCATE command committed automatically.

## UNLOAD

Unloads the result of a query to one or more text, JSON, or Apache Parquet files on Amazon S3, using Amazon S3 server-side encryption (SSE-S3). You can also specify server-side encryption with an AWS Key Management Service key (SSE-KMS) or client-side encryption with a customer managed key.

By default, the format of the unloaded file is pipe-delimited ( | ) text.

You can manage the size of files on Amazon S3, and by extension the number of files, by setting the MAXFILESIZE parameter. Ensure that the S3 IP ranges are added to your allow list. To learn more about the required S3 IP ranges, see [Network isolation](#).

You can unload the result of an Amazon Redshift query to your Amazon S3 data lake in Apache Parquet, an efficient open columnar storage format for analytics. Parquet format is up to 2x faster to unload and consumes up to 6x less storage in Amazon S3, compared with text formats. This enables you to save data transformation and enrichment you have done in Amazon S3 into your Amazon S3 data lake in an open format. You can then analyze your data with Redshift Spectrum and other AWS services such as Amazon Athena, Amazon EMR, and Amazon SageMaker.

For more information and example scenarios about using the UNLOAD command, see [Unloading data \(p. 144\)](#).

## Required privileges and permissions

For the UNLOAD command to succeed, at least SELECT privilege on the data in the database is needed, along with permission to write to the Amazon S3 location. The permissions needed are similar to the COPY command. For information about COPY command permissions, see [Permissions to access other AWS Resources \(p. 714\)](#).

## Syntax

```

UNLOAD ('select-statement')
TO 's3://object-path/name-prefix'
authorization
[option, ...]

where authorization is
IAM_ROLE { default | 'arn:aws:iam::<AWS account-id-1>:role/<role-name>[,arn:aws:iam::<AWS account-id-2>:role/<role-name>][,...]' }

where option is
| [FORMAT [AS]] CSV | PARQUET | JSON
| PARTITION BY (column_name [, ...]) [INCLUDE]
| MANIFEST [VERBOSE]
| HEADER
| DELIMITER [AS] 'delimiter-char'
| FIXEDWIDTH [AS] 'fixedwidth-spec'
| ENCRYPTED [AUTO]
| BZIP2
| GZIP
| ZSTD
| ADDQUOTES
| NULL [AS] 'null-string'
| ESCAPE
| ALLOWOVERWRITE
| CLEANPATH
| PARALLEL [{ ON | TRUE } | { OFF | FALSE }]
| MAXFILESIZE [AS] max-size [MB | GB]
| ROWGROUPSIZE [AS] size [MB | GB]
| REGION [AS] 'aws-region' }
| EXTENSION 'extension-name'
```

## Parameters

*('select-statement')*

A SELECT query. The results of the query are unloaded. In most cases, it is worthwhile to unload data in sorted order by specifying an ORDER BY clause in the query. This approach saves the time required to sort the data when it is reloaded.

The query must be enclosed in single quotation marks as shown following:

```
('select * from venue order by venueid')
```

### Note

If your query contains quotation marks (for example to enclose literal values), put the literal between two sets of single quotation marks—you must also enclose the query between single quotation marks:

```
('select * from venue where venuestate=' 'NV' ')
```

TO 's3://*object-path*/*name-prefix*'

The full path, including bucket name, to the location on Amazon S3 where Amazon Redshift writes the output file objects, including the manifest file if MANIFEST is specified. The object names are prefixed with *name-prefix*. If you use PARTITION BY, a forward slash (/) is automatically added to the end of the *name-prefix* value if needed. For added security, UNLOAD connects to Amazon S3 using an HTTPS connection. By default, UNLOAD writes one or more files per slice. UNLOAD appends a slice number and part number to the specified name prefix as follows:

*<object-path>/<name-prefix><slice-number>\_part\_<part-number>.*

If MANIFEST is specified, the manifest file is written as follows:

*<object\_path>/<name\_prefix>manifest.*

UNLOAD automatically creates encrypted files using Amazon S3 server-side encryption (SSE), including the manifest file if MANIFEST is used. The COPY command automatically reads server-side encrypted files during the load operation. You can transparently download server-side encrypted files from your bucket using either the Amazon S3 console or API. For more information, see [Protecting Data Using Server-Side Encryption](#).

To use Amazon S3 client-side encryption, specify the ENCRYPTED option.

**Important**

REGION is required when the Amazon S3 bucket isn't in the same AWS Region as the Amazon Redshift database.

*authorization*

The UNLOAD command needs authorization to write data to Amazon S3. The UNLOAD command uses the same parameters the COPY command uses for authorization. For more information, see [Authorization parameters \(p. 693\)](#) in the COPY command syntax reference.

IAM\_ROLE { default | 'arn:aws:iam:<AWS account-id-1>:role/<role-name>'}

Use the default keyword to have Amazon Redshift use the IAM role that is set as default and associated with the cluster when the UNLOAD command runs.

Use the Amazon Resource Name (ARN) for an IAM role that your cluster uses for authentication and authorization. If you specify IAM\_ROLE, you can't use ACCESS\_KEY\_ID and SECRET\_ACCESS\_KEY, SESSION\_TOKEN, or CREDENTIALS. The IAM\_ROLE can be chained. For more information, see [Chaining IAM roles](#) in the *Amazon Redshift Management Guide*.

[ FORMAT [AS] ] CSV | PARQUET | JSON

Keywords to specify the unload format to override the default format.

When CSV, unloads to a text file in CSV format using a comma ( , ) character as the default delimiter. If a field contains delimiters, double quotation marks, newline characters, or carriage returns, then the field in the unloaded file is enclosed in double quotation marks. A double quotation mark within a data field is escaped by an additional double quotation mark. When zero rows are unloaded, Amazon Redshift might write empty Amazon S3 objects.

When PARQUET, unloads to a file in Apache Parquet version 1.0 format. By default, each row group is compressed using SNAPPY compression. For more information about Apache Parquet format, see [Parquet](#).

When JSON, unloads to a JSON file with each line containing a JSON object, representing a full record in the query result. Amazon Redshift supports writing nested JSON when the query result contains SUPER columns. To create a valid JSON object, the name of each column in the query must be unique. In the JSON file, boolean values are unloaded as t or f, and NULL values are unloaded as null. When zero rows are unloaded, Amazon Redshift does not write Amazon S3 objects.

The FORMAT and AS keywords are optional. You can't use CSV with FIXEDWIDTH or ADDQUOTES. You can't use PARQUET with DELIMITER, FIXEDWIDTH, ADDQUOTES, ESCAPE, NULL AS, HEADER, GZIP, BZIP2, or ZSTD. PARQUET with ENCRYPTED is only supported with server-side encryption with an AWS Key Management Service key (SSE-KMS). You can't use JSON with DELIMITER, HEADER, FIXEDWIDTH, ADDQUOTES, ESCAPE, or NULL AS.

**PARTITION BY ( *column\_name* [, ... ] ) [INCLUDE]**

Specifies the partition keys for the unload operation. UNLOAD automatically partitions output files into partition folders based on the partition key values, following the Apache Hive convention. For example, a Parquet file that belongs to the partition year 2019 and the month September has the following prefix: s3://my\_bucket\_name/my\_prefix/year=2019/month=September/000.parquet.

The value for *column\_name* must be a column in the query results being unloaded.

If you specify PARTITION BY with the INCLUDE option, partition columns aren't removed from the unloaded files.

Amazon Redshift doesn't support string literals in PARTITION BY clauses.

**MANIFEST [ VERBOSE ]**

Creates a manifest file that explicitly lists details for the data files that are created by the UNLOAD process. The manifest is a text file in JSON format that lists the URL of each file that was written to Amazon S3.

If MANIFEST is specified with the VERBOSE option, the manifest includes the following details:

- The column names and data types, and for CHAR, VARCHAR, or NUMERIC data types, dimensions for each column. For CHAR and VARCHAR data types, the dimension is the length. For a DECIMAL or NUMERIC data type, the dimensions are precision and scale.
- The row count unloaded to each file. If the HEADER option is specified, the row count includes the header line.
- The total file size of all files unloaded and the total row count unloaded to all files. If the HEADER option is specified, the row count includes the header lines.
- The author. Author is always "Amazon Redshift".

You can specify VERBOSE only following MANIFEST.

The manifest file is written to the same Amazon S3 path prefix as the unload files in the format <object\_path\_prefix>.manifest. For example, if UNLOAD specifies the Amazon S3 path prefix 's3://mybucket/venue\_', the manifest file location is 's3://mybucket/venue\_.manifest'.

**HEADER**

Adds a header line containing column names at the top of each output file. Text transformation options, such as CSV, DELIMITER, ADDQUOTES, and ESCAPE, also apply to the header line. You can't use HEADER with FIXEDWIDTH.

**DELIMITER AS '*delimiter\_character*'**

Specifies a single ASCII character that is used to separate fields in the output file, such as a pipe character ( | ), a comma ( , ), or a tab ( \t ). The default delimiter for text files is a pipe character. The default delimiter for CSV files is a comma character. The AS keyword is optional. You can't use DELIMITER with FIXEDWIDTH. If the data contains the delimiter character, you need to specify the ESCAPE option to escape the delimiter, or use ADDQUOTES to enclose the data in double quotation marks. Alternatively, specify a delimiter that isn't contained in the data.

**FIXEDWIDTH '*fixedwidth\_spec*'**

Unloads the data to a file where each column width is a fixed length, rather than separated by a delimiter. The *fixedwidth\_spec* is a string that specifies the number of columns and the width of the

columns. The AS keyword is optional. Because FIXEDWIDTH doesn't truncate data, the specification for each column in the UNLOAD statement needs to be at least as long as the length of the longest entry for that column. The format for *fixedwidth\_spec* is shown below:

```
'colID1:colWidth1,colID2:colWidth2, ...'
```

You can't use FIXEDWIDTH with DELIMITER or HEADER.

#### ENCRYPTED [AUTO]

Specifies that the output files on Amazon S3 are encrypted using Amazon S3 server-side encryption or client-side encryption. If MANIFEST is specified, the manifest file is also encrypted. For more information, see [Unloading encrypted data files \(p. 146\)](#). If you don't specify the ENCRYPTED parameter, UNLOAD automatically creates encrypted files using Amazon S3 server-side encryption with AWS-managed encryption keys (SSE-S3).

For ENCRYPTED, you might want to unload to Amazon S3 using server-side encryption with an AWS KMS key (SSE-KMS). If so, use the [KMS\\_KEY\\_ID \(p. 995\)](#) parameter to provide the key ID. You can't use the [CREDENTIALS \(p. 694\)](#) parameter with the KMS\_KEY\_ID parameter. If you run an UNLOAD command for data using KMS\_KEY\_ID, you can then do a COPY operation for the same data without specifying a key.

To unload to Amazon S3 using client-side encryption with a customer-supplied symmetric key, provide the key in one of two ways. To provide the key, use the [MASTER\\_SYMMETRIC\\_KEY \(p. 995\)](#) parameter or the master\_symmetric\_key portion of a [CREDENTIALS \(p. 694\)](#) credential string. If you unload data using a root symmetric key, make sure that you supply the same key when you perform a COPY operation for the encrypted data.

UNLOAD doesn't support Amazon S3 server-side encryption with a customer-supplied key (SSE-C).

If ENCRYPTED AUTO is used, the UNLOAD command fetches the default AWS KMS encryption key on the target Amazon S3 bucket property and encrypts the files written to Amazon S3 with the AWS KMS key. If the bucket doesn't have the default AWS KMS encryption key, UNLOAD automatically creates encrypted files using Amazon Redshift server-side encryption with AWS-managed encryption keys (SSE-S3). You can't use this option with KMS\_KEY\_ID, MASTER\_SYMMETRIC\_KEY, or CREDENTIALS that contains master\_symmetric\_key.

#### KMS\_KEY\_ID 'key-id'

Specifies the key ID for an AWS Key Management Service (AWS KMS) key to be used to encrypt data files on Amazon S3. For more information, see [What is AWS Key Management Service?](#) If you specify KMS\_KEY\_ID, you must specify the [ENCRYPTED \(p. 995\)](#) parameter also. If you specify KMS\_KEY\_ID, you can't authenticate using the CREDENTIALS parameter. Instead, use either [IAM\\_ROLE \(p. 694\)](#) or [ACCESS\\_KEY\\_ID and SECRET\\_ACCESS\\_KEY \(p. 694\)](#).

#### MASTER\_SYMMETRIC\_KEY 'root\_key'

Specifies the root symmetric key to be used to encrypt data files on Amazon S3. If you specify MASTER\_SYMMETRIC\_KEY, you must specify the [ENCRYPTED \(p. 995\)](#) parameter also. You can't use MASTER\_SYMMETRIC\_KEY with the CREDENTIALS parameter. For more information, see [Loading encrypted data files from Amazon S3 \(p. 73\)](#).

#### BZIP2

Unloads data to one or more bzip2-compressed files per slice. Each resulting file is appended with a .bz2 extension.

#### GZIP

Unloads data to one or more gzip-compressed files per slice. Each resulting file is appended with a .gz extension.

## ZSTD

Unloads data to one or more Zstandard-compressed files per slice. Each resulting file is appended with a .zst extension.

## ADDQUOTES

Places quotation marks around each unloaded data field, so that Amazon Redshift can unload data values that contain the delimiter itself. For example, if the delimiter is a comma, you could unload and reload the following data successfully:

```
"1", "Hello, World"
```

Without the added quotation marks, the string Hello, World would be parsed as two separate fields.

Some output formats do not support ADDQUOTES.

If you use ADDQUOTES, you must specify REMOVEQUOTES in the COPY if you reload the data.

## NULL AS '*null-string*'

Specifies a string that represents a null value in unload files. If this option is used, all output files contain the specified string in place of any null values found in the selected data. If this option isn't specified, null values are unloaded as:

- Zero-length strings for delimited output
- Whitespace strings for fixed-width output

If a null string is specified for a fixed-width unload and the width of an output column is less than the width of the null string, the following behavior occurs:

- An empty field is output for non-character columns
- An error is reported for character columns

Unlike other data types where a user-defined string represents a null value, Amazon Redshift exports the SUPER data columns using the JSON format and represents it as null as determined by the JSON format. As a result, SUPER data columns ignore the NULL [AS] option used in UNLOAD commands.

## ESCAPE

For CHAR and VARCHAR columns in delimited unload files, an escape character (\) is placed before every occurrence of the following characters:

- Linefeed: \n
- Carriage return: \r
- The delimiter character specified for the unloaded data.
- The escape character: \
- A quotation mark character: " or ' (if both ESCAPE and ADDQUOTES are specified in the UNLOAD command).

### Important

If you loaded your data using a COPY with the ESCAPE option, you must also specify the ESCAPE option with your UNLOAD command to generate the reciprocal output file. Similarly, if you UNLOAD using the ESCAPE option, you need to use ESCAPE when you COPY the same data.

## ALLOWOVERWRITE

By default, UNLOAD fails if it finds files that it would possibly overwrite. If ALLOWOVERWRITE is specified, UNLOAD overwrites existing files, including the manifest file.

#### CLEANPATH

The CLEANPATH option removes existing files located in the Amazon S3 path specified in the TO clause before unloading files to the specified location.

If you include the PARTITION BY clause, existing files are removed only from the partition folders to receive new files generated by the UNLOAD operation.

You must have the s3:DeleteObject permission on the Amazon S3 bucket. For information, see [Policies and Permissions in Amazon S3](#) in the *Amazon Simple Storage Service User Guide*. Files that you remove by using the CLEANPATH option are permanently deleted and can't be recovered.

You can't specify the CLEANPATH option if you specify the ALLOWOVERWRITE option.

#### PARALLEL

By default, UNLOAD writes data in parallel to multiple files, according to the number of slices in the cluster. The default option is ON or TRUE. If PARALLEL is OFF or FALSE, UNLOAD writes to one or more data files serially, sorted absolutely according to the ORDER BY clause, if one is used. The maximum size for a data file is 6.2 GB. So, for example, if you unload 13.4 GB of data, UNLOAD creates the following three files.

|                      |        |
|----------------------|--------|
| s3://mybucket/key000 | 6.2 GB |
| s3://mybucket/key001 | 6.2 GB |
| s3://mybucket/key002 | 1.0 GB |

#### Note

The UNLOAD command is designed to use parallel processing. We recommend leaving PARALLEL enabled for most cases, especially if the files are used to load tables using a COPY command.

#### MAXFILESIZE [AS] max-size [ MB | GB ]

Specifies the maximum size of files that UNLOAD creates in Amazon S3. Specify a decimal value between 5 MB and 6.2 GB. The AS keyword is optional. The default unit is MB. If MAXFILESIZE isn't specified, the default maximum file size is 6.2 GB. The size of the manifest file, if one is used, isn't affected by MAXFILESIZE.

#### ROWGROUPSIZE [AS] size [ MB | GB ]

Specifies the size of row groups. Choosing a larger size can reduce the number of row groups, reducing the amount of network communication. Specify an integer value between 32 MB and 128 MB. The AS keyword is optional. The default unit is MB.

If ROWGROUPSIZE isn't specified, the default size is 32 MB. To use this parameter, the storage format must be Parquet and the node type must be ra3.4xlarge, ra3.16xlarge, ds2.8xlarge, or dc2.8xlarge.

#### REGION [AS] 'aws-region'

Specifies the AWS Region where the target Amazon S3 bucket is located. REGION is required for UNLOAD to an Amazon S3 bucket that isn't in the same AWS Region as the Amazon Redshift database.

The value for *aws\_region* must match an AWS Region listed in the [Amazon Redshift regions and endpoints](#) table in the *AWS General Reference*.

By default, UNLOAD assumes that the target Amazon S3 bucket is located in the same AWS Region as the Amazon Redshift database.

#### EXTENSION 'extension-name'

Specifies the file extension to append to the names of the unloaded files. Amazon Redshift doesn't run any validation, so you must verify that the specified file extension is correct. If you're using a

compression method such as GZIP, you still have to specify .gz in the extension parameter. If you don't provide any extension, Amazon Redshift doesn't add anything to the filename. If you specify a compression method without providing an extension, Amazon Redshift only adds the compression method's extension to the filename.

## Usage notes

### Using ESCAPE for all delimited text UNLOAD operations

When you UNLOAD using a delimiter, your data can include that delimiter or any of the characters listed in the ESCAPE option description. In this case, you must use the ESCAPE option with the UNLOAD statement. If you don't use the ESCAPE option with the UNLOAD, subsequent COPY operations using the unloaded data might fail.

#### Important

We strongly recommend that you always use ESCAPE with both UNLOAD and COPY statements. The exception is if you are certain that your data doesn't contain any delimiters or other characters that might need to be escaped.

### Loss of floating-point precision

You might encounter loss of precision for floating-point data that is successively unloaded and reloaded.

### Limit clause

The SELECT query can't use a LIMIT clause in the outer SELECT. For example, the following UNLOAD statement fails.

```
unload ('select * from venue limit 10')
to 's3://mybucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

Instead, use a nested LIMIT clause, as in the following example.

```
unload ('select * from venue where venueid in
(select venueid from venue order by venueid desc limit 10)')
to 's3://mybucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

You can also populate a table using SELECT...INTO or CREATE TABLE AS using a LIMIT clause, then unload from that table.

### Unloading a column of the GEOMETRY data type

You can only unload GEOMETRY columns to text or CSV format. You can't unload GEOMETRY data with the FIXEDWIDTH option. The data is unloaded in the hexadecimal form of the extended well-known binary (EWKB) format. If the size of the EWKB data is more than 4 MB, then a warning occurs because the data can't later be loaded into a table.

### Unloading the HLLSKETCH data type

You can only unload HLLSKETCH columns to text or CSV format. You can't unload HLLSKETCH data with the FIXEDWIDTH option. The data is unloaded in the Base64 format for dense HyperLogLog sketches or in the JSON format for sparse HyperLogLog sketches. For more information, see [HyperLogLog functions \(p. 1138\)](#).

The following example exports a table containing HLLSKETCH columns into a file.

```

CREATE TABLE a_table(an_int INT, b_int INT);
INSERT INTO a_table VALUES (1,1), (2,1), (3,1), (4,1), (1,2), (2,2), (3,2), (4,2), (5,2),
(6,2);

CREATE TABLE hll_table (sketch HLLSKETCH);
INSERT INTO hll_table select hll_create_sketch(an_int) from a_table group by b_int;

UNLOAD ('select * from hll_table') TO 's3://mybucket/unload/'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole' NULL AS 'null' ALLOWOVERWRITE
CSV;

```

## Unloading a column of the VARBYTE data type

You can only unload VARBYTE columns to text or CSV format. The data is unloaded in the hexadecimal form. You can't unload VARBYTE data with the FIXEDWIDTH option. The ADDQUOTES option of UNLOAD to a CSV is not supported. A VARBYTE column can't be a PARTITIONED BY column.

## FORMAT AS PARQUET clause

Be aware of these considerations when using FORMAT AS PARQUET:

- Unload to Parquet doesn't use file level compression. Each row group is compressed with SNAPPY.
- If MAXFILESIZE isn't specified, the default maximum file size is 6.2 GB. You can use MAXFILESIZE to specify a file size of 5 MB–6.2 GB. The actual file size is approximated when the file is being written, so it might not be exactly equal to the number you specify.

To maximize scan performance, Amazon Redshift tries to create Parquet files that contain equally sized 32-MB row groups. The MAXFILESIZE value that you specify is automatically rounded down to the nearest multiple of 32 MB. For example, if you specify MAXFILESIZE 200 MB, then each Parquet file unloaded is approximately 192 MB (32 MB row group x 6 = 192 MB).

- If a column uses TIMESTAMPTZ data format, only the timestamp values are unloaded. The time zone information isn't unloaded.
- Don't specify file name prefixes that begin with underscore (\_) or period (.) characters. Redshift Spectrum treats files that begin with these characters as hidden files and ignores them.

## PARTITION BY clause

Be aware of these considerations when using PARTITION BY:

- Partition columns aren't included in the output file.
- Make sure to include partition columns in the SELECT query used in the UNLOAD statement. You can specify any number of partition columns in the UNLOAD command. However, there is a limitation that there should be at least one nonpartition column to be part of the file.
- If the partition key value is null, Amazon Redshift automatically unloads that data into a default partition called `partition_column=__HIVE_DEFAULT_PARTITION__`.
- The UNLOAD command doesn't make any calls to an external catalog. To register your new partitions to be part of your existing external table, use a separate `ALTER TABLE ... ADD PARTITION ...` command. Or you can run a `CREATE EXTERNAL TABLE` command to register the unloaded data as a new external table. You can also use an AWS Glue crawler to populate your Data Catalog. For more information, see [Defining Crawlers](#) in the [AWS Glue Developer Guide](#).
- If you use the MANIFEST option, Amazon Redshift generates only one manifest file in the root Amazon S3 folder.
- The column data types that you can use as the partition key are SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, BOOLEAN, CHAR, VARCHAR, DATE, and TIMESTAMP.

## Using the ASSUMEROLE privilege to grant access to an IAM role for UNLOAD operations

To provide access for specific users and groups to an IAM role for UNLOAD operations, a superuser can grant the ASSUMEROLE privilege on an IAM role to users and groups. For information, see [GRANT \(p. 887\)](#).

### UNLOAD doesn't support Amazon S3 access point aliases

You can't use Amazon S3 access point aliases with the UNLOAD command.

## Examples

For examples that show how to use the UNLOAD command, see [UNLOAD examples \(p. 1000\)](#).

## UNLOAD examples

### Note

These examples contain line breaks for readability. Do not include line breaks or spaces in your *credentials-args* string.

### Unload VENUE to a pipe-delimited file (default delimiter)

The following example unloads the VENUE table and writes the data to s3://mybucket/unload/:

```
unload ('select * from venue')
to 's3://mybucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

By default, UNLOAD writes one or more files per slice. Assuming a two-node cluster with two slices per node, the previous example creates these files in mybucket:

```
unload/0000_part_00
unload/0001_part_00
unload/0002_part_00
unload/0003_part_00
```

To better differentiate the output files, you can include a prefix in the location. The following example unloads the VENUE table and writes the data to s3://mybucket/unload/venue\_pipe\_:

```
unload ('select * from venue')
to 's3://mybucket/unload/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

The result is these four files in the unload folder, again assuming four slices.

```
venue_pipe_0000_part_00
venue_pipe_0001_part_00
venue_pipe_0002_part_00
venue_pipe_0003_part_00
```

### Unload LINEITEM table to partitioned Parquet files

The following example unloads the LINEITEM table in Parquet format, partitioned by the l\_shipdate column.

```
unload ('select * from lineitem')
to 's3://mybucket/lineitem/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
PARQUET
PARTITION BY (l_shipdate);
```

Assuming four slices, the resulting Parquet files are dynamically partitioned into various folders.

```
s3://mybucket/lineitem/l_shipdate=1992-01-02/0000_part_00.parquet
 0001_part_00.parquet
 0002_part_00.parquet
 0003_part_00.parquet
s3://mybucket/lineitem/l_shipdate=1992-01-03/0000_part_00.parquet
 0001_part_00.parquet
 0002_part_00.parquet
 0003_part_00.parquet
s3://mybucket/lineitem/l_shipdate=1992-01-04/0000_part_00.parquet
 0001_part_00.parquet
 0002_part_00.parquet
 0003_part_00.parquet
...
...
```

#### Note

In some cases, the UNLOAD command used the INCLUDE option as shown in the following SQL statement.

```
unload ('select * from lineitem')
to 's3://mybucket/lineitem/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
PARQUET
PARTITION BY (l_shipdate) INCLUDE;
```

In these cases, the l\_shipdate column is also in the data in the Parquet files. Otherwise, the l\_shipdate column data isn't in the Parquet files.

## Unload the VENUE table to a JSON file

The following example unloads the VENUE table and writes the data in JSON format to s3://mybucket/unload/.

```
unload ('select * from venue')
to 's3://mybucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
JSON;
```

Following are sample rows from the VENUE table.

| venueid | venuename                  | venuecity   | venuestate | venueseats |
|---------|----------------------------|-------------|------------|------------|
| 1       | Pinewood Racetrack         | Akron       | OH         | 0          |
| 2       | Columbus "Crew" Stadium    | Columbus    | OH         | 0          |
| 4       | Community, Ballpark, Arena | Kansas City | KS         | 0          |

After unloading to JSON, the format of the file is similar to the following.

```
{"venueid":1,"venuename":"Pinewood
Racetrack","venuecity":"Akron","venuestate":"OH","venueseats":0}
```

```
{"venueid":2,"venuename":"Columbus \"Crew\" Stadium"
 ", "venuecity":"Columbus", "venuestate":"OH", "venueseats":0}
 {"venueid":4,"venuename":"Community, Ballpark, Arena", "venuecity":"Kansas
 City", "venuestate":"KS", "venueseats":0}
```

## Unload VENUE to a CSV file

The following example unloads the VENUE table and writes the data in CSV format to s3://mybucket/unload/.

```
unload ('select * from venue')
to 's3://mybucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
CSV;
```

Suppose that the VENUE table contains the following rows.

| venueid | venuename                  | venuecity   | venuestate | venueseats |
|---------|----------------------------|-------------|------------|------------|
| 1       | Pinewood Racetrack         | Akron       | OH         | 0          |
| 2       | Columbus "Crew" Stadium    | Columbus    | OH         | 0          |
| 4       | Community, Ballpark, Arena | Kansas City | KS         | 0          |

The unload file looks similar to the following.

```
1,Pinewood Racetrack,Akron,OH,0
2,"Columbus ""Crew"" Stadium",Columbus,OH,0
4,"Community, Ballpark, Arena",Kansas City,KS,0
```

## Unload VENUE to a CSV file using a delimiter

The following example unloads the VENUE table and writes the data in CSV format using the pipe character (|) as the delimiter. The unloaded file is written to s3://mybucket/unload/. The VENUE table in this example contains the pipe character in the value of the first row (Pinewood Race|track). It does this to show that the value in the result is enclosed in double quotation marks. A double quotation mark is escaped by a double quotation mark, and the entire field is enclosed in double quotation marks.

```
unload ('select * from venue')
to 's3://mybucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
CSV DELIMITER AS '|';
```

Suppose that the VENUE table contains the following rows.

| venueid | venuename                  | venuecity   | venuestate | venueseats |
|---------|----------------------------|-------------|------------|------------|
| 1       | Pinewood Race track        | Akron       | OH         | 0          |
| 2       | Columbus "Crew" Stadium    | Columbus    | OH         | 0          |
| 4       | Community, Ballpark, Arena | Kansas City | KS         | 0          |

The unload file looks similar to the following.

```
1|"Pinewood Race|track"|Akron|OH|0
```

```
2|"Columbus ""Crew"" Stadium"|Columbus|OH|0
4|Community, Ballpark, Arena|Kansas City|KS|0
```

## Unload VENUE with a manifest file

To create a manifest file, include the MANIFEST option. The following example unloads the VENUE table and writes a manifest file along with the data files to s3://mybucket/venue\_pipe\_:

**Important**

If you unload files with the MANIFEST option, you should use the MANIFEST option with the COPY command when you load the files. If you use the same prefix to load the files and don't specify the MANIFEST option, COPY fails because it assumes the manifest file is a data file.

```
unload ('select * from venue')
to 's3://mybucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

The result is these five files:

```
s3://mybucket/venue_pipe_0000_part_00
s3://mybucket/venue_pipe_0001_part_00
s3://mybucket/venue_pipe_0002_part_00
s3://mybucket/venue_pipe_0003_part_00
s3://mybucket/venue_pipe_manifest
```

The following shows the contents of the manifest file.

```
{
 "entries": [
 {"url": "s3://mybucket/ticket/venue_0000_part_00"},
 {"url": "s3://mybucket/ticket/venue_0001_part_00"},
 {"url": "s3://mybucket/ticket/venue_0002_part_00"},
 {"url": "s3://mybucket/ticket/venue_0003_part_00"}
]
}
```

## Unload VENUE with MANIFEST VERBOSE

When you specify the MANIFEST VERBOSE option, the manifest file includes the following sections:

- The entries section lists Amazon S3 path, file size, and row count for each file.
- The schema section lists the column names, data types, and dimension for each column.
- The meta section shows the total file size and row count for all files.

The following example unloads the VENUE table using the MANIFEST VERBOSE option.

```
unload ('select * from venue')
to 's3://mybucket/unload_venue_folder/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest verbose;
```

The following shows the contents of the manifest file.

```
{
 "entries": [
 {"url": "s3://mybucket/venue_pipe_0000_part_00", "meta": { "content_length": 32295,
 "record_count": 10 }},
```

```
{
 "url": "s3://mybucket/venue_pipe_0001_part_00", "meta": { "content_length": 32771,
 "record_count": 20 },
 {"url": "s3://mybucket/venue_pipe_0002_part_00", "meta": { "content_length": 32302,
 "record_count": 10 },
 {"url": "s3://mybucket/venue_pipe_0003_part_00", "meta": { "content_length": 31810,
 "record_count": 15 }}
],
"schema": {
 "elements": [
 {"name": "venueid", "type": { "base": "integer" }},
 {"name": "venuename", "type": { "base": "character varying", 25 }},
 {"name": "venuecity", "type": { "base": "character varying", 25 }},
 {"name": "venuestate", "type": { "base": "character varying", 25 }},
 {"name": "venueseats", "type": { "base": "character varying", 25 }}
]
},
"meta": {
 "content_length": 129178,
 "record_count": 55
},
"author": {
 "name": "Amazon Redshift",
 "version": "1.0.0"
}
}
```

## Unload VENUE with a header

The following example unloads VENUE with a header row.

```
unload ('select * from venue where venueseats > 75000')
to 's3://mybucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
header
parallel off;
```

The following shows the contents of the output file with a header row.

|                                                    |
|----------------------------------------------------|
| venueid venuename venuecity venuestate venueseats  |
| 6 New York Giants Stadium East Rutherford NJ 80242 |
| 78 INVESCO Field Denver CO 76125                   |
| 83 FedExField Landover MD 91704                    |
| 79 Arrowhead Stadium Kansas City MO 79451          |

## Unload VENUE to smaller files

By default, the maximum file size is 6.2 GB. If the unload data is larger than 6.2 GB, UNLOAD creates a new file for each 6.2 GB data segment. To create smaller files, include the MAXFILESIZE parameter. Assuming the size of the data in the previous example was 20 GB, the following UNLOAD command creates 20 files, each 1 GB in size.

```
unload ('select * from venue')
to 's3://mybucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
maxfilesize 1 gb;
```

## Unload VENUE serially

To unload serially, specify PARALLEL OFF. UNLOAD then writes one file at a time, up to a maximum of 6.2 GB per file.

The following example unloads the VENUE table and writes the data serially to s3://mybucket/unload/.

```
unload ('select * from venue')
to 's3://mybucket/unload/venue_serial_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off;
```

The result is one file named venue\_serial\_000.

If the unload data is larger than 6.2 GB, UNLOAD creates a new file for each 6.2 GB data segment. The following example unloads the LINEORDER table and writes the data serially to s3://mybucket/unload/.

```
unload ('select * from lineorder')
to 's3://mybucket/unload/lineorder_serial_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off gzip;
```

The result is the following series of files.

```
lineorder_serial_0000.gz
lineorder_serial_0001.gz
lineorder_serial_0002.gz
lineorder_serial_0003.gz
```

To better differentiate the output files, you can include a prefix in the location. The following example unloads the VENUE table and writes the data to s3://mybucket/venue\_pipe\_:

```
unload ('select * from venue')
to 's3://mybucket/unload/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

The result is these four files in the unload folder, again assuming four slices.

```
venue_pipe_0000_part_00
venue_pipe_0001_part_00
venue_pipe_0002_part_00
venue_pipe_0003_part_00
```

## Load VENUE from unload files

To load a table from a set of unload files, simply reverse the process by using a COPY command. The following example creates a new table, LOADVENUE, and loads the table from the data files created in the previous example.

```
create table loadvenue (like venue);

copy loadvenue from 's3://mybucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

If you used the MANIFEST option to create a manifest file with your unload files, you can load the data using the same manifest file. You do so with a COPY command with the MANIFEST option. The following example loads data using a manifest file.

```
copy loadvenue
```

```
from 's3://mybucket/venue_pipe_manifest' iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

## Unload VENUE to encrypted files

The following example unloads the VENUE table to a set of encrypted files using an AWS KMS key. If you specify a manifest file with the ENCRYPTED option, the manifest file is also encrypted. For more information, see [Unloading encrypted data files \(p. 146\)](#).

```
unload ('select * from venue')
to 's3://mybucket/venue_encrypt_kms'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
kms_key_id '1234abcd-12ab-34cd-56ef-1234567890ab'
manifest
encrypted;
```

The following example unloads the VENUE table to a set of encrypted files using a root symmetric key.

```
unload ('select * from venue')
to 's3://mybucket/venue_encrypt_cmk'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key 'EXAMPLEMASTERKEYtjk/OpCwtYSx/M4/t7DMCDIK722'
encrypted;
```

## Load VENUE from encrypted files

To load tables from a set of files that were created by using UNLOAD with the ENCRYPT option, reverse the process by using a COPY command. With that command, use the ENCRYPTED option and specify the same root symmetric key that was used for the UNLOAD command. The following example loads the LOADVENUE table from the encrypted data files created in the previous example.

```
create table loadvenue (like venue);

copy loadvenue
from 's3://mybucket/venue_encrypt_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key 'EXAMPLEMASTERKEYtjk/OpCwtYSx/M4/t7DMCDIK722'
manifest
encrypted;
```

## Unload VENUE data to a tab-delimited file

```
unload ('select venueid, venuename, venueseats from venue')
to 's3://mybucket/venue_tab_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter as '\t';
```

The output data files look like this:

```
1 Toyota Park Bridgeview IL 0
2 Columbus Crew Stadium Columbus OH 0
3 RFK Stadium Washington DC 0
4 CommunityAmerica Ballpark Kansas City KS 0
5 Gillette Stadium Foxborough MA 68756
...
```

## Unload VENUE to a fixed-width data file

```
unload ('select * from venue')
to 's3://mybucket/venue_fw_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
fixedwidth as 'venueid:3,venuename:39,venuecity:16,venuestate:2,venueseats:6';
```

The output data files look like the following.

```
1 Toyota Park Bridgeview IL0
2 Columbus Crew Stadium Columbus OH0
3 RFK Stadium Washington DC0
4 CommunityAmerica BallparkKansas City KS0
5 Gillette Stadium Foxborough MA68756
...
```

## Unload VENUE to a set of tab-delimited GZIP-compressed files

```
unload ('select * from venue')
to 's3://mybucket/venue_tab_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter as '\t'
gzip;
```

## Unload VENUE to a GZIP-compressed text file

```
unload ('select * from venue')
to 's3://mybucket/venue_tab_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
extension 'txt.gz'
gzip;
```

## Unload data that contains a delimiter

This example uses the ADDQUOTES option to unload comma-delimited data where some of the actual data fields contain a comma.

First, create a table that contains quotation marks.

```
create table location (id int, location char(64));
insert into location values (1,'Phoenix, AZ'),(2,'San Diego, CA'),(3,'Chicago, IL');
```

Then, unload the data using the ADDQUOTES option.

```
unload ('select id, location from location')
to 's3://mybucket/location_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter ',' addquotes;
```

The unloaded data files look like this:

```
1,"Phoenix, AZ"
2,"San Diego, CA"
3,"Chicago, IL"
...
```

## Unload the results of a join query

The following example unloads the results of a join query that contains a window function.

```
unload ('select venuecity, venuestate, caldate, pricepaid,
sum(pricepaid) over(partition by venuecity, venuestate
order by caldate rows between 3 preceding and 3 following) as winsum
from sales join date on sales.dateid=date.dateid
join event on event.eventid=sales.eventid
join venue on event.venueid=venue.venueid
order by 1,2')
to 's3://mybucket/ticket/winsum'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

The output files look like this:

```
Atlanta|GA|2008-01-04|363.00|1362.00
Atlanta|GA|2008-01-05|233.00|2030.00
Atlanta|GA|2008-01-06|310.00|3135.00
Atlanta|GA|2008-01-08|166.00|8338.00
Atlanta|GA|2008-01-11|268.00|7630.00
...
```

## Unload using NULL AS

UNLOAD outputs null values as empty strings by default. The following examples show how to use NULL AS to substitute a text string for nulls.

For these examples, we add some null values to the VENUE table.

```
update venue set venuestate = NULL
where venuecity = 'Cleveland';
```

Select from VENUE where VENUESTATE is null to verify that the columns contain NULL.

| select * from venue where venuestate is null; |                          |           |            |            |
|-----------------------------------------------|--------------------------|-----------|------------|------------|
| venueid                                       | venuename                | venuecity | venuestate | venueseats |
| 22                                            | Quicken Loans Arena      | Cleveland |            | 0          |
| 101                                           | Progressive Field        | Cleveland |            | 43345      |
| 72                                            | Cleveland Browns Stadium | Cleveland |            | 73200      |

Now, UNLOAD the VENUE table using the NULL AS option to replace null values with the character string 'fred'.

```
unload ('select * from venue')
to 's3://mybucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
null as 'fred';
```

The following sample from the unload file shows that null values were replaced with fred. It turns out that some values for VENUESEATS were also null and were replaced with fred. Even though the data type for VENUESEATS is integer, UNLOAD converts the values to text in the unload files, and then COPY converts them back to integer. If you are unloading to a fixed-width file, the NULL AS string must not be larger than the field width.

```
248|Charles Playhouse|Boston|MA|0
```

```

251|Paris Hotel|Las Vegas|NV|fred
258|Tropicana Hotel|Las Vegas|NV|fred
300|Kennedy Center Opera House|Washington|DC|0
306|Lyric Opera House|Baltimore|MD|0
308|Metropolitan Opera|New York City|NY|0
 5|Gillette Stadium|Foxborough|MA|5
 22|Quicken Loans Arena|Cleveland|fred|0
101|Progressive Field|Cleveland|fred|43345
...

```

To load a table from the unload files, use a COPY command with the same NULL AS option.

**Note**

If you attempt to load nulls into a column defined as NOT NULL, the COPY command fails.

```

create table loadvenuenulls (like venue);

copy loadvenuenulls from 's3://mybucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
null as 'fred';

```

To verify that the columns contain null, not just empty strings, select from LOADVENUENULLS and filter for null.

```

select * from loadvenuenulls where venuestate is null or venueseats is null;

venueid | venuename | venuecity | venuestate | venueseats
-----+-----+-----+-----+-----+
 72 | Cleveland Browns Stadium | Cleveland | | 73200
 253 | Mirage Hotel | Las Vegas | NV |
 255 | Venetian Hotel | Las Vegas | NV |
 22 | Quicken Loans Arena | Cleveland | | 0
 101 | Progressive Field | Cleveland | | 43345
 251 | Paris Hotel | Las Vegas | NV |
...

```

You can UNLOAD a table that contains nulls using the default NULL AS behavior and then COPY the data back into a table using the default NULL AS behavior; however, any non-numeric fields in the target table contain empty strings, not nulls. By default UNLOAD converts nulls to empty strings (white space or zero-length). COPY converts empty strings to NULL for numeric columns, but inserts empty strings into non-numeric columns. The following example shows how to perform an UNLOAD followed by a COPY using the default NULL AS behavior.

```

unload ('select * from venue')
to 's3://mybucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole' allowoverwrite;

truncate loadvenuenulls;
copy loadvenuenulls from 's3://mybucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';

```

In this case, when you filter for nulls, only the rows where VENUESEATS contained nulls. Where VENUESTATE contained nulls in the table (VENUE), VENUESTATE in the target table (LOADVENUENULLS) contains empty strings.

```

select * from loadvenuenulls where venuestate is null or venueseats is null;

venueid | venuename | venuecity | venuestate | venueseats
-----+-----+-----+-----+-----+

```

|     |                |           |    |  |
|-----|----------------|-----------|----|--|
| 253 | Mirage Hotel   | Las Vegas | NV |  |
| 255 | Venetian Hotel | Las Vegas | NV |  |
| 251 | Paris Hotel    | Las Vegas | NV |  |
| ... |                |           |    |  |

To load empty strings to non-numeric columns as NULL, include the EMPTYASNULL or BLANKSASNULL options. It's OK to use both.

```
unload ('select * from venue')
to 's3://mybucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole' allowoverwrite;

truncate loadvenuenulls;
copy loadvenuenulls from 's3://mybucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole' EMPTYASNULL;
```

To verify that the columns contain NULL, not just whitespace or empty strings, select from LOADVENUENULLS and filter for null.

```
select * from loadvenuenulls where venuestate is null or venueseats is null;
```

| venueid | venuename                | venuecity | venuestate | venueseats |
|---------|--------------------------|-----------|------------|------------|
| 72      | Cleveland Browns Stadium | Cleveland |            | 73200      |
| 253     | Mirage Hotel             | Las Vegas | NV         |            |
| 255     | Venetian Hotel           | Las Vegas | NV         |            |
| 22      | Quicken Loans Arena      | Cleveland |            | 0          |
| 101     | Progressive Field        | Cleveland |            | 43345      |
| 251     | Paris Hotel              | Las Vegas | NV         |            |
| ...     |                          |           |            |            |

## ALLOWOVERWRITE parameter example

By default, UNLOAD doesn't overwrite existing files in the destination bucket. For example, if you run the same UNLOAD statement twice without modifying the files in the destination bucket, the second UNLOAD fails. To overwrite the existing files, including the manifest file, specify the ALLOWOVERWRITE option.

```
unload ('select * from venue')
to 's3://mybucket/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest allowoverwrite;
```

# UPDATE

## Topics

- [Syntax \(p. 1011\)](#)
- [Parameters \(p. 1011\)](#)
- [Usage notes \(p. 1011\)](#)
- [Examples of UPDATE statements \(p. 1012\)](#)

Updates values in one or more table columns when a condition is satisfied.

### Note

The maximum size for a single SQL statement is 16 MB.

## Syntax

```
[WITH [RECURSIVE] common_table_expression [, common_table_expression , ...]]
 UPDATE table_name [[AS] alias] SET column = { expression | DEFAULT } [,...]
[FROM fromlist]
[WHERE condition]
```

## Parameters

### WITH clause

Optional clause that specifies one or more *common-table-expressions*. See [WITH clause \(p. 929\)](#).

*table\_name*

A temporary or persistent table. Only the owner of the table or a user with UPDATE privilege on the table may update rows. If you use the FROM clause or select from tables in an expression or condition, you must have SELECT privilege on those tables. You can't give the table an alias here; however, you can specify an alias in the FROM clause.

#### Note

Amazon Redshift Spectrum external tables are read-only. You can't UPDATE an external table.

### alias

Temporary alternative name for a target table. Aliases are optional. The AS keyword is always optional.

### SET *column* =

One or more columns that you want to modify. Columns that aren't listed retain their current values. Do not include the table name in the specification of a target column. For example, UPDATE tab SET tab.col = 1 is invalid.

### *expression*

An expression that defines the new value for the specified column.

### DEFAULT

Updates the column with the default value that was assigned to the column in the CREATE TABLE statement.

### FROM *tablelist*

You can update a table by referencing information in other tables. List these other tables in the FROM clause or use a subquery as part of the WHERE condition. Tables listed in the FROM clause can have aliases. If you need to include the target table of the UPDATE statement in the list, use an alias.

### WHERE *condition*

Optional clause that restricts updates to rows that match a condition. When the condition returns true, the specified SET columns are updated. The condition can be a simple predicate on a column or a condition based on the result of a subquery.

You can name any table in the subquery, including the target table for the UPDATE.

## Usage notes

After updating a large number of rows in a table:

- Vacuum the table to reclaim storage space and re-sort rows.
- Analyze the table to update statistics for the query planner.

Left, right, and full outer joins aren't supported in the FROM clause of an UPDATE statement; they return the following error:

ERROR: Target table must be part of an equijoin predicate

If you need to specify an outer join, use a subquery in the WHERE clause of the UPDATE statement.

If your UPDATE statement requires a self-join to the target table, you need to specify the join condition, as well as the WHERE clause criteria that qualify rows for the update operation. In general, when the target table is joined to itself or another table, a best practice is to use a subquery that clearly separates the join conditions from the criteria that qualify rows for updates.

UPDATE queries with multiple matches per row throw an error when the configuration parameter `error_on_nondeterministic_update` is set to `true`. For more information, see [error\\_on\\_nondeterministic\\_update \(p. 1692\)](#).

You can update a GENERATED BY DEFAULT AS IDENTITY column. Columns defined as GENERATED BY DEFAULT AS IDENTITY can be updated with values you supply. For more information, see [GENERATED BY DEFAULT AS IDENTITY \(p. 826\)](#).

## Examples of UPDATE statements

For more information about the tables used in the following examples, see [Sample database \(p. 1704\)](#).

The CATEGORY table in the TICKIT database contains the following rows:

| catid     | catgroup | catname   | catdesc                                    |
|-----------|----------|-----------|--------------------------------------------|
| 1         | Sports   | MLB       | Major League Baseball                      |
| 2         | Sports   | NHL       | National Hockey League                     |
| 3         | Sports   | NFL       | National Football League                   |
| 4         | Sports   | NBA       | National Basketball Association            |
| 5         | Sports   | MLS       | Major League Soccer                        |
| 6         | Shows    | Musicals  | Musical theatre                            |
| 7         | Shows    | Plays     | All non-musical theatre                    |
| 8         | Shows    | Opera     | All opera and light opera                  |
| 9         | Concerts | Pop       | All rock and pop music concerts            |
| 10        | Concerts | Jazz      | All jazz singers and bands                 |
| 11        | Concerts | Classical | All symphony, concerto, and choir concerts |
| (11 rows) |          |           |                                            |

### Updating a table based on a range of values

Update the CATGROUP column based on a range of values in the CATID column.

```
update category
set catgroup='Theatre'
where catid between 6 and 8;
```

```
select * from category
where catid between 6 and 8;
```

| catid | catgroup | catname | catdesc                 |
|-------|----------|---------|-------------------------|
| 6     | Theatre  | Plays   | All non-musical theatre |
| 7     | Theatre  | Plays   | All non-musical theatre |

```

-----+-----+-----+-----+
6 | Theatre | Musicals | Musical theatre
7 | Theatre | Plays | All non-musical theatre
8 | Theatre | Opera | All opera and light opera
(3 rows)

```

### Updating a table based on a current value

Update the CATNAME and CATDESC columns based on their current CATGROUP value:

```

update category
set catdesc=default, catname='Shows'
where catgroup='Theatre';

```

```

select * from category
where catname='Shows';

catid | catgroup | catname | catdesc
-----+-----+-----+-----+
6 | Theatre | Shows |
7 | Theatre | Shows |
8 | Theatre | Shows |
(3 rows)

```

In this case, the CATDESC column was set to null because no default value was defined when the table was created.

Run the following commands to set the CATEGORY table data back to the original values:

```

truncate category;

copy category from
's3://mybucket/data/category_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';

```

### Updating a table based on the result of a WHERE clause subquery

Update the CATEGORY table based on the result of a subquery in the WHERE clause:

```

update category
set catdesc='Broadway Musical'
where category.catid in
(select category.catid from category
join event on category.catid = event.catid
join venue on venue.venueid = event.venueid
join sales on sales.eventid = event.eventid
where venuecity='New York City' and catname='Musicals');

```

View the updated table:

```

select * from category order by 1;

catid | catgroup | catname | catdesc
-----+-----+-----+-----+
1 | Sports | MLB | Major League Baseball
2 | Sports | NHL | National Hockey League
3 | Sports | NFL | National Football League

```

```

4 | Sports | NBA | National Basketball Association
5 | Sports | MLS | Major League Soccer
6 | Shows | Musicals | Broadway Musical
7 | Shows | Plays | All non-musical theatre
8 | Shows | Opera | All opera and light opera
9 | Concerts | Pop | All rock and pop music concerts
10 | Concerts | Jazz | All jazz singers and bands
11 | Concerts | Classical | All symphony, concerto, and choir concerts
(11 rows)

```

## Updating a table based on the result of a join condition

Update the original 11 rows in the CATEGORY table based on matching CATID rows in the EVENT table:

```

update category set catid=100
from event
where event.catid=category.catid;

select * from category order by 1;
catid | catgroup | catname | catdesc
-----+-----+-----+-----+
1 | Sports | MLB | Major League Baseball
2 | Sports | NHL | National Hockey League
3 | Sports | NFL | National Football League
4 | Sports | NBA | National Basketball Association
5 | Sports | MLS | Major League Soccer
10 | Concerts | Jazz | All jazz singers and bands
11 | Concerts | Classical | All symphony, concerto, and choir concerts
100 | Shows | Opera | All opera and light opera
100 | Shows | Musicals | Musical theatre
100 | Concerts | Pop | All rock and pop music concerts
100 | Shows | Plays | All non-musical theatre
(11 rows)

```

Note that the EVENT table is listed in the FROM clause and the join condition to the target table is defined in the WHERE clause. Only four rows qualified for the update. These four rows are the rows whose CATID values were originally 6, 7, 8, and 9; only those four categories are represented in the EVENT table:

```

select distinct catid from event;
catid

9
8
6
7
(4 rows)

```

Update the original 11 rows in the CATEGORY table by extending the previous example and adding another condition to the WHERE clause. Because of the restriction on the CATGROUP column, only one row qualifies for the update (although four rows qualify for the join).

```

update category set catid=100
from event
where event.catid=category.catid
and catgroup='Concerts';

select * from category where catid=100;
catid | catgroup | catname | catdesc
-----+-----+-----+-----+

```

```
100 | Concerts | Pop | All rock and pop music concerts
(1 row)
```

An alternative way to write this example is as follows:

```
update category set catid=100
from event join category cat on event.catid=cat.catid
where cat.catgroup='Concerts';
```

The advantage to this approach is that the join criteria are clearly separated from any other criteria that qualify rows for the update. Note the use of the alias CAT for the CATEGORY table in the FROM clause.

## Updates with outer joins in the FROM clause

The previous example showed an inner join specified in the FROM clause of an UPDATE statement. The following example returns an error because the FROM clause does not support outer joins to the target table:

```
update category set catid=100
from event left join category cat on event.catid=cat.catid
where cat.catgroup='Concerts';
ERROR: Target table must be part of an equijoin predicate
```

If the outer join is required for the UPDATE statement, you can move the outer join syntax into a subquery:

```
update category set catid=100
from
(select event.catid from event left join category cat on event.catid=cat.catid) eventcat
where category.catid=eventcat.catid
and catgroup='Concerts';
```

## VACUUM

Re-sorts rows and reclaims space in either a specified table or all tables in the current database.

### Note

Only the table owner or a superuser can effectively vacuum a table. If VACUUM is run without the necessary table privileges, the operation completes successfully but has no effect.

Amazon Redshift automatically sorts data and runs VACUUM DELETE in the background. This lessens the need to run the VACUUM command. For more information, see [Vacuuming tables \(p. 110\)](#).

By default, VACUUM skips the sort phase for any table where more than 95 percent of the table's rows are already sorted. Skipping the sort phase can significantly improve VACUUM performance. To change the default sort or delete threshold for a single table, include the table name and the TO *threshold* PERCENT parameter when you run VACUUM.

Users can access tables while they are being vacuumed. You can perform queries and write operations while a table is being vacuumed, but when data manipulation language (DML) commands and a vacuum run concurrently, both might take longer. If you run UPDATE and DELETE statements during a vacuum, system performance might be reduced. VACUUM DELETE temporarily blocks update and delete operations.

Amazon Redshift automatically performs a DELETE ONLY vacuum in the background. Automatic vacuum operation pauses when users run data definition language (DDL) operations, such as ALTER TABLE.

### Note

The Amazon Redshift VACUUM command syntax and behavior are substantially different from the PostgreSQL VACUUM operation. For example, the default VACUUM operation in Amazon Redshift is VACUUM FULL, which reclaims disk space and re-sorts all rows. In contrast, the default VACUUM operation in PostgreSQL simply reclaims space and makes it available for reuse.

For more information, see [Vacuuming tables \(p. 110\)](#).

## Required privileges

Following are required privileges for VACUUM:

- Superuser
- Users with the VACUUM privilege
- Table owner
- Database owner whom the table is shared to

## Syntax

```
VACUUM [FULL | SORT ONLY | DELETE ONLY | REINDEX | RECLUSTER]
[[table_name] [TO threshold PERCENT] [BOOST]]
```

## Parameters

### FULL

Sorts the specified table (or all tables in the current database) and reclaims disk space occupied by rows that were marked for deletion by previous UPDATE and DELETE operations. VACUUM FULL is the default.

A full vacuum doesn't perform a reindex for interleaved tables. To reindex interleaved tables followed by a full vacuum, use the [VACUUM REINDEX \(p. 1017\)](#) option.

By default, VACUUM FULL skips the sort phase for any table that is already at least 95 percent sorted. If VACUUM is able to skip the sort phase, it performs a DELETE ONLY and reclaims space in the delete phase such that at least 95 percent of the remaining rows aren't marked for deletion.

If the sort threshold isn't met (for example, if 90 percent of rows are sorted) and VACUUM performs a full sort, then it also performs a complete delete operation, recovering space from 100 percent of deleted rows.

You can change the default vacuum threshold only for a single table. To change the default vacuum threshold for a single table, include the table name and the TO *threshold* PERCENT parameter.

### SORT ONLY

Sorts the specified table (or all tables in the current database) without reclaiming space freed by deleted rows. This option is useful when reclaiming disk space isn't important but re-sorting new rows is important. A SORT ONLY vacuum reduces the elapsed time for vacuum operations when the unsorted region doesn't contain a large number of deleted rows and doesn't span the entire sorted region. Applications that don't have disk space constraints but do depend on query optimizations associated with keeping table rows sorted can benefit from this kind of vacuum.

By default, VACUUM SORT ONLY skips any table that is already at least 95 percent sorted. To change the default sort threshold for a single table, include the table name and the TO *threshold* PERCENT parameter when you run VACUUM.

#### DELETE ONLY

Amazon Redshift automatically performs a DELETE ONLY vacuum in the background, so you rarely, if ever, need to run a DELETE ONLY vacuum.

A VACUUM DELETE reclaims disk space occupied by rows that were marked for deletion by previous UPDATE and DELETE operations, and compacts the table to free up the consumed space. A DELETE ONLY vacuum operation doesn't sort table data.

This option reduces the elapsed time for vacuum operations when reclaiming disk space is important but re-sorting new rows isn't important. This option can also be useful when your query performance is already optimal, and re-sorting rows to optimize query performance isn't a requirement.

By default, VACUUM DELETE ONLY reclaims space such that at least 95 percent of the remaining rows aren't marked for deletion. To change the default delete threshold for a single table, include the table name and the TO *threshold* PERCENT parameter when you run VACUUM.

Some operations, such as ALTER TABLE APPEND, can cause tables to be fragmented. When you use the DELETE ONLY clause the vacuum operation reclaims space from fragmented tables. The same threshold value of 95 percent applies to the defragmentation operation.

#### REINDEX *tablename*

Analyzes the distribution of the values in interleaved sort key columns, then performs a full VACUUM operation. If REINDEX is used, a table name is required.

VACUUM REINDEX takes significantly longer than VACUUM FULL because it makes an additional pass to analyze the interleaved sort keys. The sort and merge operation can take longer for interleaved tables because the interleaved sort might need to rearrange more rows than a compound sort.

If a VACUUM REINDEX operation terminates before it completes, the next VACUUM resumes the reindex operation before performing the full vacuum operation.

VACUUM REINDEX isn't supported with TO *threshold* PERCENT.

#### *table\_name*

The name of a table to vacuum. If you don't specify a table name, the vacuum operation applies to all tables in the current database. You can specify any permanent or temporary user-created table. The command isn't meaningful for other objects, such as views and system tables.

If you include the TO *threshold* PERCENT parameter, a table name is required.

#### RECLUSTER *tablename*

Sorts the portions of the table that are unsorted. Portions of the table that are already sorted by automatic table sort are left intact. This command doesn't merge the newly sorted data with the sorted region. It also doesn't reclaim all space that is marked for deletion. When this command completes, the table might not appear fully sorted, as indicated by the unsorted field in SVV\_TABLE\_INFO.

We recommend that you use VACUUM RECLUSTER for large tables with frequent ingestion and queries that access only the most recent data.

VACUUM RECLUSTER isn't supported with TO *threshold* PERCENT. If RECLUSTER is used, a table name is required.

VACUUM RECLUSTER isn't supported on tables with interleaved sort keys and tables with ALL distribution style.

*table\_name*

The name of a table to vacuum. You can specify any permanent or temporary user-created table. The command isn't meaningful for other objects, such as views and system tables.

TO *threshold* PERCENT

A clause that specifies the threshold above which VACUUM skips the sort phase and the target threshold for reclaiming space in the delete phase. The *sort threshold* is the percentage of total rows that are already in sort order for the specified table prior to vacuuming. The *delete threshold* is the minimum percentage of total rows not marked for deletion after vacuuming.

Because VACUUM re-sorts the rows only when the percent of sorted rows in a table is less than the sort threshold, Amazon Redshift can often reduce VACUUM times significantly. Similarly, when VACUUM isn't constrained to reclaim space from 100 percent of rows marked for deletion, it is often able to skip rewriting blocks that contain only a few deleted rows.

For example, if you specify 75 for *threshold*, VACUUM skips the sort phase if 75 percent or more of the table's rows are already in sort order. For the delete phase, VACUUM sets a target of reclaiming disk space such that at least 75 percent of the table's rows aren't marked for deletion following the vacuum. The *threshold* value must be an integer between 0 and 100. The default is 95. If you specify a value of 100, VACUUM always sorts the table unless it's already fully sorted and reclaims space from all rows marked for deletion. If you specify a value of 0, VACUUM never sorts the table and never reclaims space.

If you include the TO *threshold* PERCENT parameter, you must also specify a table name. If a table name is omitted, VACUUM fails.

You can't use the TO *threshold* PERCENT parameter with REINDEX.

BOOST

Runs the VACUUM command with additional resources, such as memory and disk space, as they're available. With the BOOST option, VACUUM operates in one window and blocks concurrent deletes and updates for the duration of the VACUUM operation. Running with the BOOST option contends for system resources, which might affect query performance. Run the VACUUM BOOST when the load on the system is light, such as during maintenance operations.

Consider the following when using the BOOST option:

- When BOOST is specified, the *table\_name* value is required.
- BOOST isn't supported with REINDEX.
- BOOST is ignored with DELETE ONLY.

## Usage notes

For most Amazon Redshift applications, a full vacuum is recommended. For more information, see [Vacuuming tables \(p. 110\)](#).

Before running a vacuum operation, note the following behavior:

- You can't run VACUUM within a transaction block (BEGIN ... END). For more information about transactions, see [Serializable isolation \(p. 119\)](#).
- You can run only one VACUUM command on a cluster at any given time. If you attempt to run multiple vacuum operations concurrently, Amazon Redshift returns an error.
- Some amount of table growth might occur when tables are vacuumed. This behavior is expected when there are no deleted rows to reclaim or the new sort order of the table results in a lower ratio of data compression.

- During vacuum operations, some degree of query performance degradation is expected. Normal performance resumes as soon as the vacuum operation is complete.
- Concurrent write operations proceed during vacuum operations, but we don't recommend performing write operations while vacuuming. It's more efficient to complete write operations before running the vacuum. Also, any data that is written after a vacuum operation has been started can't be vacuumed by that operation. In this case, a second vacuum operation is necessary.
- A vacuum operation might not be able to start if a load or insert operation is already in progress. Vacuum operations temporarily require exclusive access to tables in order to start. This exclusive access is required briefly, so vacuum operations don't block concurrent loads and inserts for any significant period of time.
- Vacuum operations are skipped when there is no work to do for a particular table; however, there is some overhead associated with discovering that the operation can be skipped. If you know that a table is pristine or doesn't meet the vacuum threshold, don't run a vacuum operation against it.
- A DELETE ONLY vacuum operation on a small table might not reduce the number of blocks used to store the data, especially when the table has a large number of columns or the cluster uses a large number of slices per node. These vacuum operations add one block per column per slice to account for concurrent inserts into the table, and there is potential for this overhead to outweigh the reduction in block count from the reclaimed disk space. For example, if a 10-column table on an 8-node cluster occupies 1000 blocks before a vacuum, the vacuum doesn't reduce the actual block count unless more than 80 blocks of disk space are reclaimed because of deleted rows. (Each data block uses 1 MB.)

Automatic vacuum operations pause if any of the following conditions are met:

- A user runs a data definition language (DDL) operation, such as ALTER TABLE, that requires an exclusive lock on a table that automatic vacuum is currently working on.
- A user triggers VACUUM on any table in the cluster (only one VACUUM can run at a time).
- A period of high cluster load.

## Examples

Reclaim space and database and re-sort rows in all tables based on the default 95 percent vacuum threshold.

```
vacuum;
```

Reclaim space and re-sort rows in the SALES table based on the default 95 percent threshold.

```
vacuum sales;
```

Always reclaim space and re-sort rows in the SALES table.

```
vacuum sales to 100 percent;
```

Re-sort rows in the SALES table only if fewer than 75 percent of rows are already sorted.

```
vacuum sort only sales to 75 percent;
```

Reclaim space in the SALES table such that at least 75 percent of the remaining rows aren't marked for deletion following the vacuum.

```
vacuum delete only sales to 75 percent;
```

Reindex and then vacuum the LISTING table.

```
vacuum reindex listing;
```

The following command returns an error.

```
vacuum reindex listing to 75 percent;
```

## SQL functions reference

### Topics

- [Leader node-only functions \(p. 1020\)](#)
- [Compute node-only functions \(p. 1021\)](#)
- [Aggregate functions \(p. 1022\)](#)
- [Array functions \(p. 1041\)](#)
- [VARBYTE functions and operators \(p. 1044\)](#)
- [Bit-wise aggregate functions \(p. 1051\)](#)
- [Conditional expressions \(p. 1056\)](#)
- [Data type formatting functions \(p. 1066\)](#)
- [Date and time functions \(p. 1086\)](#)
- [Hash functions \(p. 1131\)](#)
- [HyperLogLog functions \(p. 1138\)](#)
- [JSON functions \(p. 1142\)](#)
- [Machine learning functions \(p. 1152\)](#)
- [Math functions \(p. 1154\)](#)
- [Object functions \(p. 1179\)](#)
- [Spatial functions \(p. 1180\)](#)
- [String functions \(p. 1277\)](#)
- [SUPER type information functions \(p. 1325\)](#)
- [Window functions \(p. 1335\)](#)
- [System administration functions \(p. 1379\)](#)
- [System information functions \(p. 1386\)](#)

Amazon Redshift supports a number of functions that are extensions to the SQL standard, as well as standard aggregate functions, scalar functions, and window functions.

### Note

Amazon Redshift is based on PostgreSQL. Amazon Redshift and PostgreSQL have a number of very important differences that you must be aware of as you design and develop your data warehouse applications. For more information about how Amazon Redshift SQL differs from PostgreSQL, see [Amazon Redshift and PostgreSQL \(p. 565\)](#).

## Leader node-only functions

Some Amazon Redshift queries are distributed and run on the compute nodes; other queries run exclusively on the leader node.

The leader node distributes SQL to the compute nodes when a query references user-created tables or system tables (tables with an STL or STV prefix and system views with an SVL or SVV prefix). A query that references only catalog tables (tables with a PG prefix, such as PG\_TABLE\_DEF) or that does not reference any tables, runs exclusively on the leader node.

Some Amazon Redshift SQL functions are supported only on the leader node and are not supported on the compute nodes. A query that uses a leader-node function must run exclusively on the leader node, not on the compute nodes, or it will return an error.

The documentation for each leader-node only function includes a note stating that the function will return an error if it references user-defined tables or Amazon Redshift system tables.

For more information, see [SQL functions supported on the leader node \(p. 564\)](#).

The following SQL functions are leader-node only functions and are not supported on the compute nodes:

#### System information functions

- CURRENT\_SCHEMA
- CURRENT\_SCHEMAS
- HAS\_DATABASE\_PRIVILEGE
- HAS\_SCHEMA\_PRIVILEGE
- HAS\_TABLE\_PRIVILEGE

The following leader-node only functions are deprecated:

#### Date functions

- AGE
- CURRENT\_TIME
- CURRENT\_TIMESTAMP
- LOCALTIME
- ISFINITE
- NOW

#### String functions

- GETBIT
- GET\_BYTE
- SET\_BIT
- SET\_BYTE
- TO\_ASCII

## Compute node-only functions

Some Amazon Redshift queries must run only on the compute nodes. If a query references a user-created table, the SQL runs on the compute nodes.

A query that references only catalog tables (tables with a PG prefix, such as PG\_TABLE\_DEF) or that does not reference any tables, runs exclusively on the leader node.

If a query that uses a compute-node function doesn't reference a user-defined table or Amazon Redshift system table returns the following error.

[Amazon](500310) Invalid operation: One or more of the used functions must be applied on at least one user created table.

The documentation for each compute-node only function includes a note stating that the function will return an error if the query doesn't reference a user-defined table or Amazon Redshift system table.

The following SQL functions are compute-node only functions:

- LISTAGG
- MEDIAN
- PERCENTILE\_CONT
- PERCENTILE\_DISC and APPROXIMATE\_PERCENTILE\_DISC

## Aggregate functions

### Topics

- [ANY\\_VALUE function \(p. 1022\)](#)
- [APPROXIMATE\\_PERCENTILE\\_DISC function \(p. 1024\)](#)
- [AVG function \(p. 1025\)](#)
- [COUNT function \(p. 1027\)](#)
- [LISTAGG function \(p. 1029\)](#)
- [MAX function \(p. 1031\)](#)
- [MEDIAN function \(p. 1032\)](#)
- [MIN function \(p. 1034\)](#)
- [PERCENTILE\\_CONT function \(p. 1035\)](#)
- [STDDEV\\_SAMP and STDDEV\\_POP functions \(p. 1037\)](#)
- [SUM function \(p. 1038\)](#)
- [VAR\\_SAMP and VAR\\_POP functions \(p. 1040\)](#)

Aggregate functions compute a single result value from a set of input values.

SELECT statements using aggregate functions can include two optional clauses: GROUP BY and HAVING. The syntax for these clauses is as follows (using the COUNT function as an example):

```
SELECT count (*) expression FROM table_reference
WHERE condition [GROUP BY expression] [HAVING condition]
```

The GROUP BY clause aggregates and groups results by the unique values in a specified column or columns. The HAVING clause restricts the results returned to rows where a particular aggregate condition is true, such as `count (*) > 1`. The HAVING clause is used in the same way as WHERE to restrict rows based on the value of a column. For an example of these additional clauses, see the [COUNT \(p. 1027\)](#).

Aggregate functions don't accept nested aggregate functions or window functions as arguments.

## ANY\_VALUE function

The ANY\_VALUE function returns any value from the input expression values nondeterministically. This function can return NULL if the input expression doesn't result in any rows being returned or contains a NULL value.

## Syntax

```
ANY_VALUE ([DISTINCT | ALL] expression)
```

## Arguments

DISTINCT | ALL

Specify either DISTINCT or ALL to return any value from the input expression values. The DISTINCT argument has no effect and is ignored.

*expression*

The target column or expression on which the function operates. The *expression* is one of the following data types:

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISION
- BOOLEAN
- CHAR
- VARCHAR
- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER
- HLLSKETCH
- GEOMETRY
- GEOGRAPHY

## Returns

Returns the same data type as *expression*.

## Usage notes

If a statement that specifies the ANY\_VALUE function for a column also includes a second column reference, the second column must appear in a GROUP BY clause or be included in an aggregate function.

## Examples

The examples use the event table that is created in [Step 4: Load sample data from Amazon S3](#) in the *Amazon Redshift Getting Started Guide*. The following example returns an instance of any dateid where the eventname is Eagles.

```
select any_value(dateid) as dateid, eventname from event where eventname ='Eagles' group by eventname;
```

Following are the results.

| dateid | eventname |
|--------|-----------|
| 1878   | Eagles    |

The following example returns an instance of any dateid where the eventname is Eagles or Cold War Kids.

```
select any_value(dateid) as dateid, eventname from event where eventname in('Eagles', 'Cold War Kids') group by eventname;
```

Following are the results.

| dateid | eventname     |
|--------|---------------|
| 1922   | Cold War Kids |
| 1878   | Eagles        |

## APPROXIMATE PERCENTILE\_DISC function

APPROXIMATE PERCENTILE\_DISC is an inverse distribution function that assumes a discrete distribution model. It takes a percentile value and a sort specification and returns an element from the given set. Approximation enables the function to run much faster, with a low relative error of around 0.5 percent.

For a given *percentile* value, APPROXIMATE PERCENTILE\_DISC uses a quantile summary algorithm to approximate the discrete percentile of the expression in the ORDER BY clause. APPROXIMATE PERCENTILE\_DISC returns the value with the smallest cumulative distribution value (with respect to the same sort specification) that is greater than or equal to *percentile*.

APPROXIMATE PERCENTILE\_DISC is a compute-node only function. The function returns an error if the query doesn't reference a user-defined table or Amazon Redshift system table.

### Syntax

```
APPROXIMATE PERCENTILE_DISC (percentile)
WITHIN GROUP (ORDER BY expr)
```

### Arguments

*percentile*

Numeric constant between 0 and 1. Nulls are ignored in the calculation.

WITHIN GROUP ( ORDER BY *expr* )

Clause that specifies numeric or date/time values to sort and compute the percentile over.

### Returns

The same data type as the ORDER BY expression in the WITHIN GROUP clause.

## Usage notes

If the APPROXIMATE PERCENTILE\_DISC statement includes a GROUP BY clause, the result set is limited. The limit varies based on node type and the number of nodes. If the limit is exceeded, the function fails and returns the following error.

```
GROUP BY limit for approximate percentile_disc exceeded.
```

If you need to evaluate more groups than the limit permits, consider using [PERCENTILE\\_CONT function \(p. 1035\)](#).

## Examples

The following example returns the number of sales, total sales, and fiftieth percentile value for the top 10 dates.

```
select top 10 date.caldate,
count(totalprice), sum(totalprice),
approximate percentile_disc(0.5)
within group (order by totalprice)
from listing
join date on listing.dateid = date.dateid
group by date.caldate
order by 3 desc;

caldate | count | sum | percentile_disc
-----+-----+-----+-----+
2008-01-07 | 658 | 2081400.00 | 2020.00
2008-01-02 | 614 | 2064840.00 | 2178.00
2008-07-22 | 593 | 1994256.00 | 2214.00
2008-01-26 | 595 | 1993188.00 | 2272.00
2008-02-24 | 655 | 1975345.00 | 2070.00
2008-02-04 | 616 | 1972491.00 | 1995.00
2008-02-14 | 628 | 1971759.00 | 2184.00
2008-09-01 | 600 | 1944976.00 | 2100.00
2008-07-29 | 597 | 1944488.00 | 2106.00
2008-07-23 | 592 | 1943265.00 | 1974.00
```

## AVG function

The AVG function returns the average (arithmetic mean) of the input expression values. The AVG function works with numeric values and ignores NULL values.

### Syntax

```
AVG ([DISTINCT | ALL] expression)
```

### Arguments

*expression*

The target column or expression that the function operates on. The *expression* is one of the following data types:

- SMALLINT
- INTEGER
- BIGINT

- NUMERIC
- DECIMAL
- REAL
- DOUBLE PRECISION
- SUPER

DISTINCT | ALL

With the argument DISTINCT, the function eliminates all duplicate values from the specified expression before calculating the average. With the argument ALL, the function retains all duplicate values from the expression for calculating the average. ALL is the default.

## Data types

The argument types supported by the AVG function are SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL, DOUBLE PRECISION, and SUPER.

The return types supported by the AVG function are:

- BIGINT for any integer type argument
- DOUBLE PRECISION for a floating point argument
- Returns the same data type as expression for any other argument type.

The default precision for an AVG function result with a NUMERIC or DECIMAL argument is 38. The scale of the result is the same as the scale of the argument. For example, an AVG of a DEC(5,2) column returns a DEC(38,2) data type.

## Examples

Find the average quantity sold per transaction from the SALES table:

```
select avg(qtysold)from sales;
avg

2
(1 row)
```

Find the average total price listed for all listings:

```
select avg(numtickets*priceperticket) as avg_total_price from listing;
avg_total_price

3034.41
(1 row)
```

Find the average price paid, grouped by month in descending order:

```
select avg(pricepaid) as avg_price, month
from sales, date
where sales.dateid = date.dateid
group by month
order by avg_price desc;
```

```
avg_price | month
-----+-----
659.34 | MAR
655.06 | APR
645.82 | JAN
643.10 | MAY
642.72 | JUN
642.37 | SEP
640.72 | OCT
640.57 | DEC
635.34 | JUL
635.24 | FEB
634.24 | NOV
632.78 | AUG
(12 rows)
```

## COUNT function

The COUNT function counts the rows defined by the expression.

The COUNT function has the following variations.

- COUNT ( \* ) counts all the rows in the target table whether they include nulls or not.
- COUNT ( *expression* ) computes the number of rows with non-NULL values in a specific column or expression.
- COUNT ( DISTINCT *expression* ) computes the number of distinct non-NULL values in a column or expression.
- APPROXIMATE COUNT DISTINCT approximates the number of distinct non-NULL values in a column or expression.

## Syntax

|                                             |
|---------------------------------------------|
| <code>COUNT( *   <i>expression</i> )</code> |
|---------------------------------------------|

|                                                             |
|-------------------------------------------------------------|
| <code>COUNT ( [ DISTINCT   ALL ] <i>expression</i> )</code> |
|-------------------------------------------------------------|

|                                                               |
|---------------------------------------------------------------|
| <code>APPROXIMATE COUNT ( DISTINCT <i>expression</i> )</code> |
|---------------------------------------------------------------|

## Arguments

*expression*

The target column or expression that the function operates on. The COUNT function supports all argument data types.

**DISTINCT | ALL**

With the argument DISTINCT, the function eliminates all duplicate values from the specified expression before doing the count. With the argument ALL, the function retains all duplicate values from the expression for counting. ALL is the default.

**APPROXIMATE**

When used with APPROXIMATE, a COUNT DISTINCT function uses a HyperLogLog algorithm to approximate the number of distinct non-NULL values in a column or expression. Queries that use the APPROXIMATE keyword run much faster, with a low relative error of around 2%. Approximation

is warranted for queries that return a large number of distinct values, in the millions or more per query, or per group, if there is a group by clause. For smaller sets of distinct values, in the thousands, approximation might be slower than a precise count. APPROXIMATE can only be used with COUNT DISTINCT.

## Return type

The COUNT function returns BIGINT.

## Examples

Count all of the users from the state of Florida:

```
select count(*) from users where state='FL';

count

510
```

Count all of the event names from the EVENT table:

```
select count(eventname) from event;

count

8798
```

Count all of the event names from the EVENT table:

```
select count(all eventname) from event;

count

8798
```

Count all of the unique venue IDs from the EVENT table:

```
select count(distinct venueid) as venues from event;

venues

204
```

Count the number of times each seller listed batches of more than four tickets for sale. Group the results by seller ID:

```
select count(*), sellerid from listing
where numtickets > 4
group by sellerid
order by 1 desc, 2;

count | sellerid
-----+-----
12 | 6386
11 | 17304
11 | 20123
11 | 25428
```

...

The following examples compare the return values and execution times for COUNT and APPROXIMATE COUNT.

```
select count(distinct pricepaid) from sales;

count

4528

Time: 48.048 ms

select approximate count(distinct pricepaid) from sales;

count

4553

Time: 21.728 ms
```

## LISTAGG function

For each group in a query, the LISTAGG aggregate function orders the rows for that group according to the ORDER BY expression, then concatenates the values into a single string.

LISTAGG is a compute-node only function. The function returns an error if the query doesn't reference a user-defined table or Amazon Redshift system table. For more information, see [Querying the catalog tables \(p. 1680\)](#).

### Syntax

```
LISTAGG([DISTINCT] aggregate_expression [, 'delimiter'])
[WITHIN GROUP (ORDER BY order_list)]
```

### Arguments

#### DISTINCT

(Optional) A clause that eliminates duplicate values from the specified expression before concatenating. Trailing spaces are ignored, so the strings 'a' and 'a ' are treated as duplicates. LISTAGG uses the first value encountered. For more information, see [Significance of trailing blanks \(p. 584\)](#).

#### *aggregate\_expression*

Any valid expression (such as a column name) that provides the values to aggregate. NULL values and empty strings are ignored.

#### *delimiter*

(Optional) The string constant to separate the concatenated values. The default is NULL.

#### *WITHIN GROUP (ORDER BY *order\_list*)*

(Optional) A clause that specifies the sort order of the aggregated values.

## Returns

VARCHAR(MAX). If the result set is larger than the maximum VARCHAR size (64K – 1, or 65535), then LISTAGG returns the following error:

```
Invalid operation: Result size exceeds LISTAGG limit
```

## Usage notes

If a statement includes multiple LISTAGG functions that use WITHIN GROUP clauses, each WITHIN GROUP clause must use the same ORDER BY values.

For example, the following statement will return an error.

```
select listagg(sellerid)
within group (order by dateid) as sellers,
listagg(dateid)
within group (order by sellerid) as dates
from winsales;
```

The following statements will run successfully.

```
select listagg(sellerid)
within group (order by dateid) as sellers,
listagg(dateid)
within group (order by dateid) as dates
from winsales;

select listagg(sellerid)
within group (order by dateid) as sellers,
listagg(dateid) as dates
from winsales;
```

## Examples

The following example aggregates seller IDs, ordered by seller ID.

```
select listagg(sellerid, ', ') within group (order by sellerid) from sales
where eventid = 4337;
listagg

380, 380, 1178, 1178, 1178, 2731, 8117, 12905, 32043, 32043, 32043, 32432, 32432, 38669,
38750, 41498, 45676, 46324, 47188, 48294
```

The following example uses DISTINCT to return a list of unique seller IDs.

```
select listagg(distinct sellerid, ', ') within group (order by sellerid) from sales
where eventid = 4337;
listagg

380, 1178, 2731, 8117, 12905, 32043, 32432, 38669, 38750, 41498, 45676, 46324, 47188, 48294
```

The following example aggregates seller IDs in date order.

```
select listagg(sellerid)
```

```
within group (order by dateid)
from winsales;

listagg

31141242333
```

The following example returns a pipe-separated list of sales dates for buyer B.

```
select listagg(dateid,'|')
within group (order by sellerid desc,salesid asc)
from winsales
where buyerid = 'b';

listagg

2003-08-02|2004-04-18|2004-04-18|2004-02-12
```

The following example returns a comma-separated list of sales IDs for each buyer ID.

```
select buyerid,
listagg(salesid,',')
within group (order by salesid) as sales_id
from winsales
group by buyerid
order by buyerid;

buyerid | sales_id
-----+-----
a |10005,40001,40005
b |20001,30001,30004,30003
c |10001,20002,30007,10006
```

## MAX function

The MAX function returns the maximum value in a set of rows. DISTINCT or ALL might be used but do not affect the result.

### Syntax

```
MAX ([DISTINCT | ALL] expression)
```

### Arguments

*expression*

The target column or expression that the function operates on. The *expression* is one of the following data types:

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISION
- CHAR

- VARCHAR
- DATE
- TIMESTAMP
- TSTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

DISTINCT | ALL

With the argument DISTINCT, the function eliminates all duplicate values from the specified expression before calculating the maximum. With the argument ALL, the function retains all duplicate values from the expression for calculating the maximum. ALL is the default.

## Data types

Returns the same data type as *expression*. The Boolean equivalent of the MIN function is the [BOOL\\_AND function \(p. 1054\)](#), and the Boolean equivalent of MAX is the [BOOL\\_OR function \(p. 1055\)](#).

## Examples

Find the highest price paid from all sales:

```
select max(pricepaid) from sales;

max

12624.00
(1 row)
```

Find the highest price paid per ticket from all sales:

```
select max(pricepaid/qtysold) as max_ticket_price
from sales;

max_ticket_price

2500.00000000
(1 row)
```

## MEDIAN function

Calculates the median value for the range of values. NULL values in the range are ignored.

MEDIAN is an inverse distribution function that assumes a continuous distribution model.

MEDIAN is a special case of [PERCENTILE\\_CONT \(p. 1035\)\(.5\)](#).

MEDIAN is a compute-node only function. The function returns an error if the query doesn't reference a user-defined table or Amazon Redshift system table.

## Syntax

```
MEDIAN (median_expression)
```

## Arguments

### *median\_expression*

The target column or expression that the function operates on.

## Data types

The return type is determined by the data type of *median\_expression*. The following table shows the return type for each *median\_expression* data type.

| Input type                         | Return type |
|------------------------------------|-------------|
| INT2, INT4, INT8, NUMERIC, DECIMAL | DECIMAL     |
| FLOAT, DOUBLE                      | DOUBLE      |
| DATE                               | DATE        |
| TIMESTAMP                          | TIMESTAMP   |
| TIMESTAMPTZ                        | TIMESTAMPTZ |

## Usage notes

If the *median\_expression* argument is a DECIMAL data type defined with the maximum precision of 38 digits, it is possible that MEDIAN will return either an inaccurate result or an error. If the return value of the MEDIAN function exceeds 38 digits, the result is truncated to fit, which causes a loss of precision. If, during interpolation, an intermediate result exceeds the maximum precision, a numeric overflow occurs and the function returns an error. To avoid these conditions, we recommend either using a data type with lower precision or casting the *median\_expression* argument to a lower precision.

If a statement includes multiple calls to sort-based aggregate functions (LISTAGG, PERCENTILE\_CONT, or MEDIAN), they must all use the same ORDER BY values. Note that MEDIAN applies an implicit order by on the expression value.

For example, the following statement returns an error.

```
select top 10 salesid, sum(pricepaid),
percentile_cont(0.6) within group (order by salesid),
median (pricepaid)
from sales group by salesid, pricepaid;

An error occurred when executing the SQL command:
select top 10 salesid, sum(pricepaid),
percentile_cont(0.6) within group (order by salesid),
median (pricepaid)
from sales group by salesid, pricepaid...

ERROR: within group ORDER BY clauses for aggregate functions must be the same
```

The following statement runs successfully.

```
select top 10 salesid, sum(pricepaid),
percentile_cont(0.6) within group (order by salesid),
median (salesid)
from sales group by salesid, pricepaid;
```

## Examples

The following example shows that MEDIAN produces the same results as PERCENTILE\_CONT(0.5).

```
select top 10 distinct sellerid, qtysold,
percentile_cont(0.5) within group (order by qtysold),
median (qtysold)
from sales
group by sellerid, qtysold;

sellerid | qtysold | percentile_cont | median
-----+-----+-----+-----
 1 | 1 | 1.0 | 1.0
 2 | 3 | 3.0 | 3.0
 5 | 2 | 2.0 | 2.0
 9 | 4 | 4.0 | 4.0
 12 | 1 | 1.0 | 1.0
 16 | 1 | 1.0 | 1.0
 19 | 2 | 2.0 | 2.0
 19 | 3 | 3.0 | 3.0
 22 | 2 | 2.0 | 2.0
 25 | 2 | 2.0 | 2.0
```

## MIN function

The MIN function returns the minimum value in a set of rows. DISTINCT or ALL might be used but do not affect the result.

### Syntax

|                                                           |
|-----------------------------------------------------------|
| <code>MIN ( [ DISTINCT   ALL ] <i>expression</i> )</code> |
|-----------------------------------------------------------|

### Arguments

*expression*

The target column or expression that the function operates on. The *expression* is one of the following data types:

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISION
- CHAR
- VARCHAR
- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

## DISTINCT | ALL

With the argument DISTINCT, the function eliminates all duplicate values from the specified expression before calculating the minimum. With the argument ALL, the function retains all duplicate values from the expression for calculating the minimum. ALL is the default.

## Data types

Returns the same data type as *expression*. The Boolean equivalent of the MIN function is [BOOL\\_AND function \(p. 1054\)](#), and the Boolean equivalent of MAX is [BOOL\\_OR function \(p. 1055\)](#).

## Examples

Find the lowest price paid from all sales:

```
select min(pricepaid) from sales;

min

20.00
(1 row)
```

Find the lowest price paid per ticket from all sales:

```
select min(pricepaid/qtysold)as min_ticket_price
from sales;

min_ticket_price

20.00000000
(1 row)
```

## PERCENTILE\_CONT function

PERCENTILE\_CONT is an inverse distribution function that assumes a continuous distribution model. It takes a percentile value and a sort specification, and returns an interpolated value that would fall into the given percentile value with respect to the sort specification.

PERCENTILE\_CONT computes a linear interpolation between values after ordering them. Using the percentile value (P) and the number of not null rows (N) in the aggregation group, the function computes the row number after ordering the rows according to the sort specification. This row number (RN) is computed according to the formula  $RN = (1 + (P * (N - 1)))$ . The final result of the aggregate function is computed by linear interpolation between the values from rows at row numbers CRN = CEILING(RN) and FRN = FLOOR(RN).

The final result will be as follows.

If (CRN = FRN = RN) then the result is (value of expression from row at RN)

Otherwise the result is as follows:

$$(CRN - RN) * (\text{value of expression for row at FRN}) + (RN - FRN) * (\text{value of expression for row at CRN}).$$

PERCENTILE\_CONT is a compute-node only function. The function returns an error if the query doesn't reference a user-defined table or Amazon Redshift system table.

## Syntax

```
PERCENTILE_CONT (percentile)
WITHIN GROUP (ORDER BY expr)
```

## Arguments

*percentile*

Numeric constant between 0 and 1. Nulls are ignored in the calculation.

WITHIN GROUP ( ORDER BY *expr* )

Specifies numeric or date/time values to sort and compute the percentile over.

## Returns

The return type is determined by the data type of the ORDER BY expression in the WITHIN GROUP clause. The following table shows the return type for each ORDER BY expression data type.

| Input type                         | Return type |
|------------------------------------|-------------|
| INT2, INT4, INT8, NUMERIC, DECIMAL | DECIMAL     |
| FLOAT, DOUBLE                      | DOUBLE      |
| DATE                               | DATE        |
| TIMESTAMP                          | TIMESTAMP   |
| TIMESTAMPTZ                        | TIMESTAMPTZ |

## Usage notes

If the ORDER BY expression is a DECIMAL data type defined with the maximum precision of 38 digits, it is possible that PERCENTILE\_CONT will return either an inaccurate result or an error. If the return value of the PERCENTILE\_CONT function exceeds 38 digits, the result is truncated to fit, which causes a loss of precision.. If, during interpolation, an intermediate result exceeds the maximum precision, a numeric overflow occurs and the function returns an error. To avoid these conditions, we recommend either using a data type with lower precision or casting the ORDER BY expression to a lower precision.

If a statement includes multiple calls to sort-based aggregate functions (LISTAGG, PERCENTILE\_CONT, or MEDIAN), they must all use the same ORDER BY values. Note that MEDIAN applies an implicit order by on the expression value.

For example, the following statement returns an error.

```
select top 10 salesid, sum(pricepaid),
percentile_cont(0.6) within group (order by salesid),
median (pricepaid)
from sales group by salesid, pricepaid;
```

An error occurred when executing the SQL command:

```
select top 10 salesid, sum(pricepaid),
percentile_cont(0.6) within group (order by salesid),
median (pricepaid)
from sales group by salesid, pricepai...
```

ERROR: within group ORDER BY clauses for aggregate functions must be the same

The following statement runs successfully.

```
select top 10 salesid, sum(pricepaid),
percentile_cont(0.6) within group (order by salesid),
median (salesid)
from sales group by salesid, pricepaid;
```

## Examples

The following example shows that MEDIAN produces the same results as PERCENTILE\_CONT(0.5).

```
select top 10 distinct sellerid, qtysold,
percentile_cont(0.5) within group (order by qtysold),
median (qtysold)
from sales
group by sellerid, qtysold;
```

| sellerid | qtysold | percentile_cont | median |
|----------|---------|-----------------|--------|
| 1        | 1       | 1.0             | 1.0    |
| 2        | 3       | 3.0             | 3.0    |
| 5        | 2       | 2.0             | 2.0    |
| 9        | 4       | 4.0             | 4.0    |
| 12       | 1       | 1.0             | 1.0    |
| 16       | 1       | 1.0             | 1.0    |
| 19       | 2       | 2.0             | 2.0    |
| 19       | 3       | 3.0             | 3.0    |
| 22       | 2       | 2.0             | 2.0    |
| 25       | 2       | 2.0             | 2.0    |

## STDDEV\_SAMP and STDDEV\_POP functions

The STDDEV\_SAMP and STDDEV\_POP functions return the sample and population standard deviation of a set of numeric values (integer, decimal, or floating-point). The result of the STDDEV\_SAMP function is equivalent to the square root of the sample variance of the same set of values.

STDDEV\_SAMP and STDDEV are synonyms for the same function.

### Syntax

```
STDDEV_SAMP | STDDEV ([DISTINCT | ALL] expression)
STDDEV_POP ([DISTINCT | ALL] expression)
```

The expression must have an integer, decimal, or floating point data type. Regardless of the data type of the expression, the return type of this function is a double precision number.

#### Note

Standard deviation is calculated using floating point arithmetic, which might result in slight imprecision.

### Usage notes

When the sample standard deviation (STDDEV or STDDEV\_SAMP) is calculated for an expression that consists of a single value, the result of the function is NULL not 0.

## Examples

The following query returns the average of the values in the VENUESEATS column of the VENUE table, followed by the sample standard deviation and population standard deviation of the same set of values. VENUESEATS is an INTEGER column. The scale of the result is reduced to 2 digits.

```
select avg(venueseats),
 cast(stddev_samp(venueseats) as dec(14,2)) stddevsamp,
 cast(stddev_pop(venueseats) as dec(14,2)) stddevpop
 from venue;

avg | stddevsamp | stddevpop
-----+-----+-----
17503 | 27847.76 | 27773.20
(1 row)
```

The following query returns the sample standard deviation for the COMMISSION column in the SALES table. COMMISSION is a DECIMAL column. The scale of the result is reduced to 10 digits.

```
select cast(stddev(commission) as dec(18,10))
 from sales;

stddev

130.3912659086
(1 row)
```

The following query casts the sample standard deviation for the COMMISSION column as an integer.

```
select cast(stddev(commission) as integer)
 from sales;

stddev

130
(1 row)
```

The following query returns both the sample standard deviation and the square root of the sample variance for the COMMISSION column. The results of these calculations are the same.

```
select
 cast(stddev_samp(commission) as dec(18,10)) stddevsamp,
 cast(sqrt(var_samp(commission)) as dec(18,10)) sqrtvarsamp
 from sales;

stddevsamp | sqrtvarsamp
-----+-----
130.3912659086 | 130.3912659086
(1 row)
```

## SUM function

The SUM function returns the sum of the input column or expression values. The SUM function works with numeric values and ignores NULL values.

### Syntax

|                                              |
|----------------------------------------------|
| SUM ( [ DISTINCT   ALL ] <i>expression</i> ) |
|----------------------------------------------|

## Arguments

### *expression*

The target column or expression that the function operates on. The *expression* is one of the following data types:

- SMALLINT
- INTEGER
- BIGINT
- NUMERIC
- DECIMAL
- REAL
- DOUBLE PRECISION
- SUPER

### DISTINCT | ALL

With the argument DISTINCT, the function eliminates all duplicate values from the specified expression before calculating the sum. With the argument ALL, the function retains all duplicate values from the expression for calculating the sum. ALL is the default.

## Data types

The argument types supported by the SUM function are SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL, DOUBLE PRECISION, and SUPER.

The return types supported by the SUM function are

- BIGINT for BIGINT, SMALLINT, and INTEGER arguments
- NUMERIC for NUMERIC arguments
- DOUBLE PRECISION for floating point arguments
- Returns the same data type as expression for any other argument type.

The default precision for a SUM function result with a NUMERIC or DECIMAL argument is 38. The scale of the result is the same as the scale of the argument. For example, a SUM of a DEC(5,2) column returns a DEC(38,2) data type.

## Examples

Find the sum of all commissions paid from the SALES table:

```
select sum(commission) from sales;

sum

16614814.65
(1 row)
```

Find the number of seats in all venues in the state of Florida:

```
select sum(venueseats) from venue
where venuestate = 'FL';

sum

```

```
250411
(1 row)
```

Find the number of seats sold in May:

```
select sum(qtysold) from sales, date
where sales.dateid = date.dateid and date.month = 'MAY';

sum

32291
(1 row)
```

## VAR\_SAMP and VAR\_POP functions

The VAR\_SAMP and VAR\_POP functions return the sample and population variance of a set of numeric values (integer, decimal, or floating-point). The result of the VAR\_SAMP function is equivalent to the squared sample standard deviation of the same set of values.

VAR\_SAMP and VARIANCE are synonyms for the same function.

### Syntax

```
VAR_SAMP | VARIANCE ([DISTINCT | ALL] expression)
VAR_POP ([DISTINCT | ALL] expression)
```

The expression must have an integer, decimal, or floating-point data type. Regardless of the data type of the expression, the return type of this function is a double precision number.

#### Note

The results of these functions might vary across data warehouse clusters, depending on the configuration of the cluster in each case.

### Usage notes

When the sample variance (VARIANCE or VAR\_SAMP) is calculated for an expression that consists of a single value, the result of the function is NULL not 0.

### Examples

The following query returns the rounded sample and population variance of the NUMTICKETS column in the LISTING table.

```
select avg(numtickets),
round(var_samp(numtickets)) varsamp,
round(var_pop(numtickets)) varpop
from listing;

avg | varsamp | varpop
-----+-----+-----
10 | 54 | 54
(1 row)
```

The following query runs the same calculations but casts the results to decimal values.

```
select avg(numtickets),
cast(var_samp(numtickets) as dec(10,4)) varsamp,
cast(var_pop(numtickets) as dec(10,4)) varpop
from listing;
```

```
avg | varsamp | varpop
-----+-----+-----
10 | 53.6291 | 53.6288
(1 row)
```

## Array functions

Following, you can find a description for the array functions for SQL that Amazon Redshift supports to access and manipulate arrays.

### Topics

- [array function \(p. 1041\)](#)
- [array\\_concat function \(p. 1042\)](#)
- [array\\_flatten function \(p. 1042\)](#)
- [get\\_array\\_length function \(p. 1043\)](#)
- [split\\_to\\_array function \(p. 1043\)](#)
- [subarray function \(p. 1044\)](#)

## array function

Creates an array of the SUPER data type.

### Syntax

```
ARRAY([expr1] [, expr2 [, ...]])
```

### Argument

*expr1, expr2*

Expressions of any Amazon Redshift data type except date and time types, since Amazon Redshift doesn't cast the date and time types to the SUPER data type. The arguments don't need to be of the same data type.

### Return type

The array function returns the SUPER data type.

### Example

The following examples show an array of numeric values and an array of different data types.

```
--an array of numeric values
select array(1,50,null,100);
 array

[1,50,null,100]
(1 row)

--an array of different data types
select array(1,'abc',true,3.14);
 array

[1,"abc",true,3.14]
```

(1 row)

## array\_concat function

The array\_concat function concatenates two arrays to create an array that contains all the elements in the first array followed by all the elements in the second array. The two arguments must be valid arrays.

### Syntax

```
array_concat(super_expr1, super_expr2)
```

### Arguments

*super\_expr1*

The value that specifies the first of the two arrays to concatenate.

*super\_expr2*

The value that specifies the second of the two arrays to concatenate.

### Return type

The array\_concat function returns a SUPER data value.

### Example

The following examples shows concatenation of two arrays of the same type and concatenation of two arrays of different types.

```
-- concatenating two arrays
SELECT ARRAY_CONCAT(ARRAY(10001,10002),ARRAY(10003,10004));
 array_concat

[10001,10002,10003,10004]
(1 row)

-- concatenating two arrays of different types
SELECT ARRAY_CONCAT(ARRAY(10001,10002),ARRAY('ab','cd'));
 array_concat

[10001,10002,"ab","cd"]
(1 row)
```

## array\_flatten function

Merges multiple arrays into a single array of SUPER type.

### Syntax

```
array_flatten(super_expr1,super_expr2,...)
```

### Arguments

*super\_expr1*,*super\_expr2*

A valid SUPER expression of array form.

## Return type

The array\_flatten function returns a SUPER data value.

## Example

The following example shows an array\_flatten function.

```
SELECT ARRAY_FLATTEN(ARRAY(ARRAY(1,2,3,4),ARRAY(5,6,7,8),ARRAY(9,10)));
array_flatten

[1,2,3,4,5,6,7,8,9,10]
(1 row)
```

## get\_array\_length function

Returns the length of the specified array. The GET\_ARRAY\_LENGTH function returns the length of a SUPER array given an object or array path.

## Syntax

```
get_array_length(super_expr)
```

## Arguments

*super\_expr*

A valid SUPER expression of array form.

## Return type

The get\_array\_length function returns a BIGINT.

## Example

The following example shows a get\_array\_length function.

```
SELECT GET_ARRAY_LENGTH(ARRAY(1,2,3,4,5,6,7,8,9,10));
get_array_length

10
(1 row)
```

## split\_to\_array function

Uses a delimiter as an optional parameter. If no delimiter is present, then the default is a comma.

## Syntax

```
split_to_array(string,delimiter)
```

## Arguments

*string*

The input string to be split.

### delimiter

An optional value on which the input string will be split. The default is a comma.

### Return type

The `split_to_array` function returns a SUPER data value.

### Example

The following example show a `split_to_array` function.

```
SELECT SPLIT_TO_ARRAY('12|345|6789', '|');
split_to_array

[\"12\", \"345\", \"6789\"]
(1 row)
```

## subarray function

Manipulates arrays to return a subset of the input arrays.

### Syntax

```
SUBARRAY(super_expr, start_position, length)
```

### Arguments

#### *super\_expr*

A valid SUPER expression in array form.

#### *start\_position*

The position within the array to begin the extraction, starting at index position 0. A negative position counts backward from the end of the array.

#### *length*

The number of elements to extract (the length of the substring).

### Return type

The `subarray` function returns a SUPER data value.

### Examples

The following is an example of a `subarray` function.

```
SELECT SUBARRAY(ARRAY('a', 'b', 'c', 'd', 'e', 'f'), 2, 3);
subarray

[\"c\", \"d\", \"e\"]
(1 row)
```

## VARBYTE functions and operators

Amazon Redshift functions and operators that support the VARBYTE data type include:

- [VARBYTE operators \(p. 1045\)](#)
- [FROM\\_HEX \(p. 1046\)](#)
- [FROM\\_VARBYTE \(p. 1047\)](#)
- [GETBIT \(p. 1048\)](#)
- [TO\\_HEX \(p. 1048\)](#)
- [TO\\_VARBYTE \(p. 1049\)](#)
- [CONCAT \(p. 1286\)](#)
- [LEN \(p. 1292\)](#)
- [LENGTH function \(p. 1293\)](#)
- [OCTET\\_LENGTH \(p. 1297\)](#)
- [SUBSTR function \(p. 1318\)](#)
- [SUBSTRING function \(p. 1318\)](#)

## VARBYTE operators

The following table lists the VARBYTE operators. The operator works with binary value of data type VARBYTE. If one or both inputs is null, the result is null.

### Supported operators

| Operator | Description           | Return type |
|----------|-----------------------|-------------|
|          | Concatenation         | BOOLEAN     |
| +        | Concatenation         | BOOLEAN     |
| <        | Less than             | BOOLEAN     |
| <=       | Less than or equal    | BOOLEAN     |
| =        | Equal                 | BOOLEAN     |
| >        | Greater than          | BOOLEAN     |
| >=       | Greater than or equal | BOOLEAN     |
| != or <> | Not equal             | BOOLEAN     |
| ~        | Bitwise not           | VARBYTE     |
| &        | Bitwise and           | VARBYTE     |
|          | Bitwise or            | VARBYTE     |
| #        | Bitwise xor           | VARBYTE     |

### Examples

In the following examples, the value of 'a' ::varbyte is 61 and the value of 'b' ::varbyte is 62.

The following example concatenates two binary values.

```
select 'a'::varbyte || 'b'::varbyte as concat;

concat

6162
```

The following example concatenates two binary values.

```
select 'a'::varbyte + 'b'::varbyte as concat;

concat

6162
```

The following example compares if 'a' is less than 'b'.

```
select 'a'::varbyte < 'b'::varbyte as less_than;

less_than

t
```

The following example compares if 'a' equals 'b'.

```
select 'a'::varbyte = 'b'::varbyte as equal;

equal

f
```

The following example negates each bit of the input binary value. Where 'a' evaluates to 01100001.

```
select from_varbyte(~'a'::varbyte, 'binary');

from_varbyte

10011110
```

The following example applies the bitwise\_and operator on the two input binary values. Where 'a' evaluates to 01100001 and 'b' evaluates to 01100010.

```
select from_varbyte('a'::varbyte & 'b'::varbyte, 'binary');

from_varbyte

01100000
```

## FROM\_HEX function

FROM\_HEX converts a hexadecimal to a binary value.

### Syntax

```
from_hex(hex_string)
```

## Arguments

### *hex\_string*

Hexadecimal string of data type VARCHAR or TEXT to be converted. The format must be a literal value.

## Return type

VARBYTE

## Examples

The following example converts the hexadecimal representation of '6162' to a binary value. The result is automatically shown as the hexadecimal representation of the binary value.

```
select from_hex('6162');

from_hex

6162
```

## FROM\_VARBYTE function

FROM\_VARBYTE converts a binary value to a character string in the specified format.

## Syntax

```
from_varbyte(binary_value, format)
```

## Arguments

### *binary\_value*

A binary value of data type VARBYTE.

### *format*

The format of the returned character string. Case insensitive valid values are hex, binary, utf-8, and utf8.

## Return type

VARCHAR

## Examples

The following example converts the binary value 'ab' to hexadecimal.

```
select from_varbyte('ab', 'hex');

from_varbyte

6162
```

The following example returns the binary representation of '4d'. The binary representation of '4d' is the character string 01001101.

```
select from_varbyte(from_hex('4d'), 'binary');

from_varbyte

01001101
```

## GETBIT function

GETBIT returns the bit value of a binary value at the specified by an index.

### Syntax

```
getbit(binary_value, index)
```

### Arguments

*binary\_value*

A binary value of data type VARBYTE.

*index*

An index number of the bit in the binary value that is returned. The binary value is a 0-based bit array that is indexed from the rightmost bit (least significant bit) to the leftmost bit (most significant bit).

### Return type

INTEGER

### Examples

The following example returns the bit at index 2 of the binary value `from_hex('4d')`. The binary representation of '4d' is 01001101.

```
select getbit(from_hex('4d'), 2);

getbit

1
```

The following example returns the bit at eight index locations of the binary value returned by `from_hex('4d')`. The binary representation of '4d' is 01001101.

```
select getbit(from_hex('4d'), 7), getbit(from_hex('4d'), 6),
 getbit(from_hex('4d'), 5), getbit(from_hex('4d'), 4),
 getbit(from_hex('4d'), 3), getbit(from_hex('4d'), 2),
 getbit(from_hex('4d'), 1), getbit(from_hex('4d'), 0);

getbit | getbit | getbit | getbit | getbit | getbit | getbit |
-----+-----+-----+-----+-----+-----+-----+
 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1
```

## TO\_HEX function

TO\_HEX converts a number or binary value to a hexadecimal representation.

## Syntax

```
to_hex(value)
```

## Arguments

*value*

Either a number or binary value (VARBYTE) to be converted.

## Return type

VARCHAR

## Examples

The following example converts a number to its hexadecimal representation.

```
select to_hex(2147676847);
 to_hex

8002f2af
```

The following example converts the VARBYTE representation of 'abc' to a hexadecimal number.

```
select to_hex('abc'::varbyte);
 to_hex

616263
```

The following example creates a table, inserts the VARBYTE representation of 'abc' to a hexadecimal number, then selects the column with the value.

```
create table t (vc varchar);
insert into t select to_hex('abc'::varbyte);
select vc from t;
 vc

616263
```

The following example shows that when casting a VARBYTE value to VARCHAR the format is UTF-8.

```
create table t (vc varchar);
insert into t select 'abc'::varbyte::varchar;
select vc from t;
 vc

abc
```

## TO\_VARBYTE function

TO\_VARBYTE converts a string in a specified format to a binary value.

## Syntax

```
to_varbyte(string, format)
```

## Arguments

*string*

A character string.

*format*

The format of the input string. Case insensitive valid values are hex, binary, utf-8, and utf8.

## Return type

VARBYTE

## Examples

The following example converts the hex 6162 to a binary value. The result is automatically shown as the hexadecimal representation of the binary value.

```
select to_varbyte('6162', 'hex');
to_varbyte

6162
```

The following example returns the binary for of 4d. The binary representation of '4d' is 01001101.

```
select to_varbyte('01001101', 'binary');
to_varbyte

4d
```

The following example converts the string 'a' in UTF-8 to a binary value. The result is automatically shown as the hexadecimal representation of the binary value.

```
select to_varbyte('a', 'utf8');
to_varbyte

61
```

The following example converts the string '4' in hexadecimal to a binary value. If the hexadecimal string length is an odd number, then a 0 is prepended to form a valid hexadecimal number.

```
select to_varbyte('4', 'hex');
to_varbyte

04
```

## Bit-wise aggregate functions

Bit-wise aggregate functions compute bit operations to perform aggregation of integer columns and columns that can be converted or rounded to integer values.

### Topics

- [Using NULLs in bit-wise aggregations \(p. 1051\)](#)
- [DISTINCT support for bit-wise aggregations \(p. 1051\)](#)
- [Overview examples for bit-wise functions \(p. 1051\)](#)
- [BIT\\_AND function \(p. 1052\)](#)
- [BIT\\_OR function \(p. 1053\)](#)
- [BOOL\\_AND function \(p. 1054\)](#)
- [BOOL\\_OR function \(p. 1055\)](#)

## Using NULLs in bit-wise aggregations

When you apply a bit-wise function to a column that is nullable, any NULL values are eliminated before the function result is calculated. If no rows qualify for aggregation, the bit-wise function returns NULL. The same behavior applies to regular aggregate functions. Following is an example.

```
select sum(venueseats), bit_and(venueseats) from venue
where venueseats is null;

sum | bit_and
-----+-----
null | null
(1 row)
```

## DISTINCT support for bit-wise aggregations

As other aggregate functions do, bit-wise functions support the DISTINCT keyword.

However, using DISTINCT with these functions has no impact on the results. The first instance of a value is sufficient to satisfy bit-wise AND or OR operations. It makes no difference if duplicate values are present in the expression being evaluated.

Because the DISTINCT processing is likely to incur some query execution overhead, we recommend that you don't use DISTINCT with bit-wise functions.

## Overview examples for bit-wise functions

Following, you can find some overview examples demonstrating how to work with the bit-wise functions. You can also find specific code examples with each function description.

Examples for the bit-wise functions are based on the TICKIT sample database. The USERS table in the TICKIT sample database contains several Boolean columns that indicate whether each user is known to like different types of events, such as sports, theatre, opera, and so on. An example follows.

```
select userid, username, lastname, city, state,
likesports, liketheatre
from users limit 10;

userid | username | lastname | city | state | likesports | liketheatre
-----+-----+-----+-----+-----+-----+-----+
1 | JSG99FHE | Taylor | Kent | WA | t | t
```

|   |  |          |  |         |  |             |  |    |  |   |  |   |
|---|--|----------|--|---------|--|-------------|--|----|--|---|--|---|
| 9 |  | MSD36KVR |  | Watkins |  | Port Orford |  | MD |  | t |  | f |
|---|--|----------|--|---------|--|-------------|--|----|--|---|--|---|

Assume that a new version of the USERS table is built in a different way. In this new version, a single integer column that defines (in binary form) eight types of events that each user likes or dislikes. In this design, each bit position represents a type of event. A user who likes all eight types has all eight bits set to 1 (as in the first row of the following table). A user who doesn't like any of these events has all eight bits set to 0 (see the second row). A user who likes only sports and jazz is represented in the third row following.

|        | <b>SPORTS</b> | <b>THEATRE</b> | <b>JAZZ</b> | <b>OPERA</b> | <b>ROCK</b> | <b>VEGAS</b> | <b>BROADWA</b> | <b>CLASSICAL</b> |
|--------|---------------|----------------|-------------|--------------|-------------|--------------|----------------|------------------|
| User 1 | 1             | 1              | 1           | 1            | 1           | 1            | 1              | 1                |
| User 2 | 0             | 0              | 0           | 0            | 0           | 0            | 0              | 0                |
| User 3 | 1             | 0              | 1           | 0            | 0           | 0            | 0              | 0                |

In the database table, these binary values can be stored in a single LIKES column as integers, as shown following.

| <b>User</b> | <b>Binary value</b> | <b>Stored value (integer)</b> |
|-------------|---------------------|-------------------------------|
| User 1      | 11111111            | 255                           |
| User 2      | 00000000            | 0                             |
| User 3      | 10100000            | 160                           |

## BIT\_AND function

The BIT\_AND function runs bit-wise AND operations on all of the values in a single integer column or expression. This function aggregates each bit of each binary value that corresponds to each integer value in the expression.

The BIT\_AND function returns a result of 0 if none of the bits is set to 1 across all of the values. If one or more bits is set to 1 across all values, the function returns an integer value. This integer is the number that corresponds to the binary value for those bits.

For example, a table contains four integer values in a column: 3, 7, 10, and 22. These integers are represented in binary form as follows:

| <b>Integer</b> | <b>Binary value</b> |
|----------------|---------------------|
| 3              | 11                  |
| 7              | 111                 |
| 10             | 1010                |
| 22             | 10110               |

A BIT\_AND operation on this dataset finds that all bits are set to 1 in the second-to-last position only. The result is a binary value of 00000010, which represents the integer value 2. Therefore, the BIT\_AND function returns 2.

## Syntax

```
BIT_AND ([DISTINCT | ALL] expression)
```

## Arguments

### *expression*

The target column or expression that the function operates on. This expression must have an INT, INT2, or INT8 data type. The function returns an equivalent INT, INT2, or INT8 data type.

### DISTINCT | ALL

With the argument DISTINCT, the function eliminates all duplicate values for the specified expression before calculating the result. With the argument ALL, the function retains all duplicate values. ALL is the default. For more information, see [DISTINCT support for bit-wise aggregations \(p. 1051\)](#).

## Examples

Given that meaningful business information is stored in integer columns, you can use bit-wise functions to extract and aggregate that information. The following query applies the BIT\_AND function to the LIKES column in a table called USERLIKES and groups the results by the CITY column.

```
select city, bit_and(likes) from userlikes group by city
order by city;
city | bit_and
-----+-----
Los Angeles | 0
Sacramento | 0
San Francisco | 0
San Jose | 64
Santa Barbara | 192
(5 rows)
```

You can interpret these results as follows:

- The integer value 192 for Santa Barbara translates to the binary value 11000000. In other words, all users in this city like sports and theatre, but not all users like any other type of event.
- The integer 64 translates to 01000000. So, for users in San Jose, the only type of event that they all like is theatre.
- The values of 0 for the other three cities indicate that no "likes" are shared by all users in those cities.

## BIT\_OR function

The BIT\_OR function runs bit-wise OR operations on all of the values in a single integer column or expression. This function aggregates each bit of each binary value that corresponds to each integer value in the expression.

For example, suppose that your table contains four integer values in a column: 3, 7, 10, and 22. These integers are represented in binary form as follows.

| Integer | Binary value |
|---------|--------------|
| 3       | 11           |

| Integer | Binary value |
|---------|--------------|
| 7       | 111          |
| 10      | 1010         |
| 22      | 10110        |

If you apply the BIT\_OR function to the set of integer values, the operation looks for any value in which a 1 is found in each position. In this case, a 1 exists in the last five positions for at least one of the values, yielding a binary result of 00011111; therefore, the function returns 31 (or 16 + 8 + 4 + 2 + 1).

## Syntax

```
BIT_OR ([DISTINCT | ALL] expression)
```

### Arguments

#### *expression*

The target column or expression that the function operates on. This expression must have an INT, INT2, or INT8 data type. The function returns an equivalent INT, INT2, or INT8 data type.

#### DISTINCT | ALL

With the argument DISTINCT, the function eliminates all duplicate values for the specified expression before calculating the result. With the argument ALL, the function retains all duplicate values. ALL is the default. For more information, see [DISTINCT support for bit-wise aggregations \(p. 1051\)](#).

## Example

The following query applies the BIT\_OR function to the LIKES column in a table called USERLIKES and groups the results by the CITY column.

```
select city, bit_or(likes) from userlikes group by city
order by city;
city | bit_or
-----+-----
Los Angeles | 127
Sacramento | 255
San Francisco | 255
San Jose | 255
Santa Barbara | 255
(5 rows)
```

For four of the cities listed, all of the event types are liked by at least one user ( $255=11111111$ ). For Los Angeles, all of the event types except sports are liked by at least one user ( $127=01111111$ ).

## BOOL\_AND function

The BOOL\_AND function operates on a single Boolean or integer column or expression. This function applies similar logic to the BIT\_AND and BIT\_OR functions. For this function, the return type is a Boolean value (true or false).

If all values in a set are true, the BOOL\_AND function returns true (t). If any value is false, the function returns false (f).

## Syntax

```
BOOL_AND ([DISTINCT | ALL] expression)
```

## Arguments

*expression*

The target column or expression that the function operates on. This expression must have a BOOLEAN or integer data type. The return type of the function is BOOLEAN.

DISTINCT | ALL

With the argument DISTINCT, the function eliminates all duplicate values for the specified expression before calculating the result. With the argument ALL, the function retains all duplicate values. ALL is the default. For more information, see [DISTINCT support for bit-wise aggregations \(p. 1051\)](#).

## Examples

You can use the Boolean functions against either Boolean expressions or integer expressions. For example, the following query return results from the standard USERS table in the TICKIT database, which has several Boolean columns.

The BOOL\_AND function returns false for all five rows. Not all users in each of those states likes sports.

```
select state, bool_and(likesports) from users
group by state order by state limit 5;

state | bool_and
-----+-----
AB | f
AK | f
AL | f
AZ | f
BC | f
(5 rows)
```

## BOOL\_OR function

The BOOL\_OR function operates on a single Boolean or integer column or expression. This function applies similar logic to the BIT\_AND and BIT\_OR functions. For this function, the return type is a Boolean value (true or false).

If any value in a set is true, the BOOL\_OR function returns true (t). If no value in a set is true, the function returns false (f).

## Syntax

```
BOOL_OR ([DISTINCT | ALL] expression)
```

## Arguments

*expression*

The target column or expression that the function operates on. This expression must have a BOOLEAN or integer data type. The return type of the function is BOOLEAN.

## DISTINCT | ALL

With the argument DISTINCT, the function eliminates all duplicate values for the specified expression before calculating the result. With the argument ALL, the function retains all duplicate values. ALL is the default. See [DISTINCT support for bit-wise aggregations \(p. 1051\)](#).

## Examples

You can use the Boolean functions with either Boolean expressions or integer expressions. For example, the following query return results from the standard USERS table in the TICKIT database, which has several Boolean columns.

The BOOL\_OR function returns true for all five rows. At least one user in each of those states likes sports.

```
select state, bool_or(likesports) from users
group by state order by state limit 5;

state | bool_or
-----+-----
AB | t
AK | t
AL | t
AZ | t
BC | t
(5 rows)
```

# Conditional expressions

## Topics

- [CASE conditional expression \(p. 1056\)](#)
- [COALESCE function \(p. 1058\)](#)
- [DECODE function \(p. 1058\)](#)
- [GREATEST and LEAST functions \(p. 1060\)](#)
- [NVL function \(p. 1061\)](#)
- [NVL2 function \(p. 1063\)](#)
- [NULLIF function \(p. 1065\)](#)

Amazon Redshift supports some conditional expressions that are extensions to the SQL standard.

## CASE conditional expression

The CASE expression is a conditional expression, similar to if/then/else statements found in other languages. CASE is used to specify a result when there are multiple conditions. Use CASE where a SQL expression is valid, such as in a SELECT command.

There are two types of CASE expressions: simple and searched.

- In simple CASE expressions, an expression is compared with a value. When a match is found, the specified action in the THEN clause is applied. If no match is found, the action in the ELSE clause is applied.
- In searched CASE expressions, each CASE is evaluated based on a Boolean expression, and the CASE statement returns the first matching CASE. If no match is found among the WHEN clauses, the action in the ELSE clause is returned.

## Syntax

Simple CASE statement used to match conditions:

```
CASE expression
 WHEN value THEN result
 [WHEN...]
 [ELSE result]
END
```

Searched CASE statement used to evaluate each condition:

```
CASE
 WHEN condition THEN result
 [WHEN ...]
 [ELSE result]
END
```

## Arguments

*expression*

A column name or any valid expression.

*value*

Value that the expression is compared with, such as a numeric constant or a character string.

*result*

The target value or expression that is returned when an expression or Boolean condition is evaluated. The data types of all the result expressions must be convertible to a single output type.

*condition*

A Boolean expression that evaluates to true or false. If *condition* is true, the value of the CASE expression is the result that follows the condition, and the remainder of the CASE expression is not processed. If *condition* is false, any subsequent WHEN clauses are evaluated. If no WHEN condition results are true, the value of the CASE expression is the result of the ELSE clause. If the ELSE clause is omitted and no condition is true, the result is null.

## Examples

The following examples use the VENUE table and SALES table from the sample TICKIT data. For more information, see [Sample database \(p. 1704\)](#).

Use a simple CASE expression to replace New York City with Big Apple in a query against the VENUE table. Replace all other city names with other.

```
select venuecity,
 case venuecity
 when 'New York City'
 then 'Big Apple' else 'other'
 end
 from venue
 order by venueid desc;

venuecity | case
-----+-----
Los Angeles | other
```

|               |           |
|---------------|-----------|
| New York City | Big Apple |
| San Francisco | other     |
| Baltimore     | other     |
| ...           |           |

Use a searched CASE expression to assign group numbers based on the PRICEPAID value for individual ticket sales:

```
select pricepaid,
 case when pricepaid <10000 then 'group 1'
 when pricepaid >10000 then 'group 2'
 else 'group 3'
 end
 from sales
 order by 1 desc;

pricepaid | case
-----+-----
12624 | group 2
10000 | group 3
10000 | group 3
9996 | group 1
9988 | group 1
...
```

## COALESCE function

Returns the value of the first expression that isn't null in a series of expressions. COALESCE has an identical expression, which is NVL. The NVL topic contains information and examples for COALESCE. For more information, see [NVL function \(p. 1061\)](#).

## DECODE function

A DECODE expression replaces a specific value with either another specific value or a default value, depending on the result of an equality condition. This operation is equivalent to the operation of a simple CASE expression or an IF-THEN-ELSE statement.

### Syntax

|                                                                                                                                 |
|---------------------------------------------------------------------------------------------------------------------------------|
| <code>DECODE ( <i>expression</i>, <i>search</i>, <i>result</i> [, <i>search</i>, <i>result</i>]... [ ,<i>default</i> ] )</code> |
|---------------------------------------------------------------------------------------------------------------------------------|

This type of expression is useful for replacing abbreviations or codes that are stored in tables with meaningful business values that are needed for reports.

### Parameters

*expression*

The source of the value that you want to compare, such as a column in a table.

*search*

The target value that is compared against the source expression, such as a numeric value or a character string. The search expression must evaluate to a single fixed value. You cannot specify an expression that evaluates to a range of values, such as age between 20 and 29; you need to specify separate search/result pairs for each value that you want to replace.

The data type of all instances of the search expression must be the same or compatible. The *expression* and *search* parameters must also be compatible.

*result*

The replacement value that query returns when the expression matches the search value. You must include at least one search/result pair in the DECODE expression.

The data types of all instances of the result expression must be the same or compatible. The *result* and *default* parameters must also be compatible.

*default*

An optional default value that is used for cases when the search condition fails. If you do not specify a default value, the DECODE expression returns NULL.

## Usage notes

If the *expression* value and the *search* value are both NULL, the DECODE result is the corresponding *result* value. For an illustration of this use of the function, see the Examples section.

When used this way, DECODE is similar to [NVL2 function \(p. 1063\)](#), but there are some differences. For a description of these differences, see the NVL2 usage notes.

## Examples

When the value 2008-06-01 exists in the caldate column of datetable, the following example replaces it with June 1st, 2008. The example replaces all other caldate values with NULL.

```
select decode(caldate, '2008-06-01', 'June 1st, 2008')
from datetable where month='JUN' order by caldate;

case

June 1st, 2008

...
(30 rows)
```

The following example uses a DECODE expression to convert the five abbreviated CATNAME columns in the CATEGORY table to full names and convert other values in the column to Unknown.

```
select catid, decode(catname,
'NHL', 'National Hockey League',
'MLB', 'Major League Baseball',
'MLS', 'Major League Soccer',
'NFL', 'National Football League',
'NBA', 'National Basketball Association',
'Unknown')
from category
order by catid;

catid | case
-----+
1 | Major League Baseball
2 | National Hockey League
3 | National Football League
4 | National Basketball Association
5 | Major League Soccer
6 | Unknown
7 | Unknown
8 | Unknown
9 | Unknown
10 | Unknown
```

|           |         |
|-----------|---------|
| 11        | Unknown |
| (11 rows) |         |

Use a DECODE expression to find venues in Colorado and Nevada with NULL in the VENUESEATS column; convert the NULLs to zeroes. If the VENUESEATS column is not NULL, return 1 as the result.

```
select venuename, venuestate, decode(venueseats,null,0,1)
from venue
where venuestate in('NV','CO')
order by 2,3,1;

venuname | venuestate | case
-----+-----+-----+
Coors Field | CO | 1
Dick's Sporting Goods Park | CO | 1
Ellie Caulkins Opera House | CO | 1
INVESCO Field | CO | 1
Pepsi Center | CO | 1
Ballys Hotel | NV | 0
Bellagio Hotel | NV | 0
Caesars Palace | NV | 0
Harrahs Hotel | NV | 0
Hilton Hotel | NV | 0
...
(20 rows)
```

## GREATEST and LEAST functions

Returns the largest or smallest value from a list of any number of expressions.

### Syntax

|                          |
|--------------------------|
| GREATEST (value [, ...]) |
| LEAST (value [, ...])    |

### Parameters

*expression\_list*

A comma-separated list of expressions, such as column names. The expressions must all be convertible to a common data type. NULL values in the list are ignored. If all of the expressions evaluate to NULL, the result is NULL.

### Returns

Returns the greatest (for GREATEST) or least (for LEAST) value from the provided list of expressions.

### Example

The following example returns the highest value alphabetically for `firstname` or `lastname`.

```
select firstname, lastname, greatest(firstname,lastname) from users
where userid < 10
order by 3;

firstname | lastname | greatest
-----+-----+-----+
```

|          |           |          |
|----------|-----------|----------|
| Lars     | Ratliff   | Ratliff  |
| Reagan   | Hodge     | Reagan   |
| Colton   | Roy       | Roy      |
| Barry    | Roy       | Roy      |
| Tamekah  | Juarez    | Tamekah  |
| Rafael   | Taylor    | Taylor   |
| Victor   | Hernandez | Victor   |
| Vladimir | Humphrey  | Vladimir |
| Mufutau  | Watkins   | Watkins  |
| (9 rows) |           |          |

## NVL function

Returns the value of the first expression that isn't null in a series of expressions. When a non-null value is found, the remaining expressions in the list aren't evaluated.

NVL is identical to COALESCE. They are synonyms. This topic explains the syntax and contains examples for both.

### Syntax

```
NVL(expression, expression, ...)
```

The syntax for COALESCE is the same:

```
COALESCE(expression, expression, ...)
```

If all expressions are null, the result is null.

These functions are useful when you want to return a secondary value when a primary value is missing or null. For example, a query might return the first of three available phone numbers: cell, home, or work. The order of the expressions in the function determines the order of evaluation.

### Arguments

*expression*

An expression, such as a column name, to be evaluated for null status.

### Return type

Amazon Redshift determines the data type of the returned value based on the input expressions. If the data types of the input expressions don't have a common type, then an error is returned.

### Examples

If the list contains integer expressions, the function returns an integer.

```
SELECT COALESCE(NULL, 12, NULL);
coalesce

12
```

The following example returns a string type.

```
SELECT COALESCE(NULL, 'Amazon Redshift', NULL);

coalesce

Amazon Redshift
```

The following example results in an error because the data types vary in the expression list. In this case, there is both a string type and a number type in the list.

```
SELECT COALESCE(NULL, 'Amazon Redshift', 12);
ERROR: invalid input syntax for integer: "Amazon Redshift"
```

For this example, you create a table with START\_DATE and END\_DATE columns, insert rows that include null values, then apply an NVL expression to the two columns.

```
create table datetable (start_date date, end_date date);
insert into datetable values ('2008-06-01','2008-12-31');
insert into datetable values (null,'2008-12-31');
insert into datetable values ('2008-12-31',null);
```

```
select nvl(start_date, end_date)
from datetable
order by 1;

coalesce

2008-06-01
2008-12-31
2008-12-31
```

The default column name for an NVL expression is COALESCE. The following query returns the same results:

```
select coalesce(start_date, end_date)
from datetable
order by 1;
```

For the following example queries, you create a table with sample hotel-booking information and insert several rows. Some records contain null values.

```
create table booking_info (booking_id int, booking_code character(8), check_in date,
check_out date, funds_collected numeric(12,2));
```

Insert the following sample data. Some records don't have a check\_out date or funds\_collected amount.

```
insert into booking_info values (1, 'OCEAN_WV', '2023-02-01','2023-02-03',100.00);
insert into booking_info values (2, 'OCEAN_WV', '2023-04-22','2023-04-26',120.00);
insert into booking_info values (3, 'DSRT_SUN', '2023-03-13','2023-03-16',125.00);
insert into booking_info values (4, 'DSRT_SUN', '2023-06-01','2023-06-03',140.00);
insert into booking_info values (5, 'DSRT_SUN', '2023-07-10',null,null);
insert into booking_info values (6, 'OCEAN_WV', '2023-08-15',null,null);
```

The following query returns a list of dates. If the check\_out date isn't available, it lists the check\_in date.

```
select coalesce(check_out, check_in)
from booking_info
order by booking_id;
```

The results are the following. Note that the last two records show the check\_in date.

```
coalesce

2023-02-03
2023-04-26
2023-03-16
2023-06-03
2023-07-10
2023-08-15
```

If you expect a query to return null values for certain functions or columns, you can use an NVL expression to replace the nulls with some other value. For example, aggregate functions, such as SUM, return null values instead of zeroes when they have no rows to evaluate. You can use an NVL expression to replace these null values with 700.0. Instead of 485, the result of summing the funds\_collected is 1885 because two rows that have null are replaced with 700.

```
select sum(nvl(funds_collected, 700.0)) as sumresult from booking_info;
sumresult

1885
```

## NVL2 function

Returns one of two values based on whether a specified expression evaluates to NULL or NOT NULL.

### Syntax

```
NVL2 (expression, not_null_return_value, null_return_value)
```

### Arguments

#### *expression*

An expression, such as a column name, to be evaluated for null status.

#### *not\_null\_return\_value*

The value returned if *expression* evaluates to NOT NULL. The *not\_null\_return\_value* value must either have the same data type as *expression* or be implicitly convertible to that data type.

#### *null\_return\_value*

The value returned if *expression* evaluates to NULL. The *null\_return\_value* value must either have the same data type as *expression* or be implicitly convertible to that data type.

### Return type

The NVL2 return type is determined as follows:

- If either *not\_null\_return\_value* or *null\_return\_value* is null, the data type of the not-null expression is returned.

If both *not\_null\_return\_value* and *null\_return\_value* are not null:

- If *not\_null\_return\_value* and *null\_return\_value* have the same data type, that data type is returned.
- If *not\_null\_return\_value* and *null\_return\_value* have different numeric data types, the smallest compatible numeric data type is returned.
- If *not\_null\_return\_value* and *null\_return\_value* have different datetime data types, a timestamp data type is returned.
- If *not\_null\_return\_value* and *null\_return\_value* have different character data types, the data type of *not\_null\_return\_value* is returned.
- If *not\_null\_return\_value* and *null\_return\_value* have mixed numeric and non-numeric data types, the data type of *not\_null\_return\_value* is returned.

### **Important**

In the last two cases where the data type of *not\_null\_return\_value* is returned, *null\_return\_value* is implicitly cast to that data type. If the data types are incompatible, the function fails.

### Usage notes

[DECODE function \(p. 1058\)](#) can be used in a similar way to NVL2 when the *expression* and *search* parameters are both null. The difference is that for DECODE, the return will have both the value and the data type of the *result* parameter. In contrast, for NVL2, the return will have the value of either the *not\_null\_return\_value* or *null\_return\_value* parameter, whichever is selected by the function, but will have the data type of *not\_null\_return\_value*.

For example, assuming column1 is NULL, the following queries will return the same value. However, the DECODE return value data type will be INTEGER and the NVL2 return value data type will be VARCHAR.

```
select decode(column1, null, 1234, '2345');
select nvl2(column1, '2345', 1234);
```

### Example

The following example modifies some sample data, then evaluates two fields to provide appropriate contact information for users:

```
update users set email = null where firstname = 'Aphrodite' and lastname = 'Acevedo';

select (firstname + ' ' + lastname) as name,
nvl2(email, email, phone) AS contact_info
from users
where state = 'WA'
and lastname like 'A%'
order by lastname, firstname;

name contact_info
-----+-----
Aphrodite Acevedo (906) 632-4407
Caldwell Acevedo Nunc.sollicitudin@Duisac.ca
Quinn Adams vel@adipiscingligulaAenean.com
Kamal Aguilar quis@vulputaterisusa.com
Samson Alexander hendrerit.neque@indolorFusce.ca
Hall Alford ac.mattis@vitaediamProin.edu
Lane Allen et.netus@risusDonec.org
Xander Allison ac.facilisis.facilisis@Infaucibus.com
Amaya Alvarado dui.nec.tempus@eudui.edu
Vera Alvarez at.arcu.Vestibulum@pellentesque.edu
Yetta Anthony enim.sit@risus.org
Violet Arnold ad.litora@at.com
```

```
August Ashley consectetuer.euismod@Phasellus.com
Karyn Austin ipsum.primis.in@Maurisblandit enim.org
Lucas Ayers at@elitpretiumet.com
```

## NULLIF function

### Syntax

The NULLIF expression compares two arguments and returns null if the arguments are equal. If they are not equal, the first argument is returned. This expression is the inverse of the NVL or COALESCE expression.

```
NULLIF (expression1, expression2)
```

### Arguments

*expression1, expression2*

The target columns or expressions that are compared. The return type is the same as the type of the first expression. The default column name of the NULLIF result is the column name of the first expression.

### Examples

In the following example, the query returns null when the LISTID and SALESID values match:

```
select nullif(listid,salesid), salesid
from sales where salesid<10 order by 1, 2 desc;

listid | salesid
-----+-----
 4 | 2
 5 | 4
 5 | 3
 6 | 5
 10 | 9
 10 | 8
 10 | 7
 10 | 6
 | 1
(9 rows)
```

You can use NULLIF to ensure that empty strings are always returned as nulls. In the example below, the NULLIF expression returns either a null value or a string that contains at least one character.

```
insert into category
values(0,'','Special','Special');

select nullif(catgroup,'') from category
where catdesc='Special';

catgroup

null
(1 row)
```

NULLIF ignores trailing spaces. If a string is not empty but contains spaces, NULLIF still returns null:

```
create table nulliftest(c1 char(2), c2 char(2));

insert into nulliftest values ('a','a ');

insert into nulliftest values ('b','b');

select nullif(c1,c2) from nulliftest;
c1

null
null
(2 rows)
```

## Data type formatting functions

### Topics

- [CAST function \(p. 1066\)](#)
- [CONVERT function \(p. 1069\)](#)
- [TO\\_CHAR \(p. 1071\)](#)
- [TO\\_DATE function \(p. 1075\)](#)
- [TO\\_NUMBER \(p. 1076\)](#)
- [TEXT\\_TO\\_INT\\_ALT \(p. 1077\)](#)
- [TEXT\\_TO\\_NUMERIC\\_ALT \(p. 1079\)](#)
- [Datetime format strings \(p. 1080\)](#)
- [Numeric format strings \(p. 1082\)](#)
- [Teradata-style formatting characters for numeric data \(p. 1083\)](#)

Data type formatting functions provide an easy way to convert values from one data type to another. For each of these functions, the first argument is always the value to be formatted and the second argument contains the template for the new format. Amazon Redshift supports several data type formatting functions.

## CAST function

The CAST function converts one data type to another compatible data type. For instance, you can convert a string to a date, or a numeric type to a string. CAST performs a runtime conversion, which means that the conversion doesn't change a value's data type in a source table. It's changed only in the context of the query.

The CAST function is very similar to [the section called "CONVERT" \(p. 1069\)](#), in that they both convert one data type to another, but they are called differently.

Certain data types require an explicit conversion to other data types using the CAST or CONVERT function. Other data types can be converted implicitly, as part of another command, without using CAST or CONVERT. See [Type compatibility and conversion \(p. 597\)](#).

### Syntax

Use either of these two equivalent syntax forms to cast expressions from one data type to another.

```
CAST (expression AS type)
expression ::= type
```

## Arguments

*expression*

An expression that evaluates to one or more values, such as a column name or a literal. Converting null values returns nulls. The expression cannot contain blank or empty strings.

*type*

One of the supported [Data types \(p. 574\)](#).

## Return type

CAST returns the data type specified by the *type* argument.

### Note

Amazon Redshift returns an error if you try to perform a problematic conversion, such as a DECIMAL conversion that loses precision, like the following:

```
select 123.456::decimal(2,1);
```

or an INTEGER conversion that causes an overflow:

```
select 12345678::smallint;
```

## Examples

Some of the examples use the sample [TICKIT database](#). For more information about setting up sample data, see [Getting started with Amazon Redshift clusters and data loading](#).

The following two queries are equivalent. They both cast a decimal value to an integer:

```
select cast(pricepaid as integer)
from sales where salesid=100;

pricepaid

162
(1 row)
```

```
select pricepaid::integer
from sales where salesid=100;

pricepaid

162
(1 row)
```

The following produces a similar result. It doesn't require sample data to run:

```
select cast(162.00 as integer) as pricepaid;

pricepaid

162
(1 row)
```

In this example, the values in a timestamp column are cast as dates, which results in removing the time from each result:

```
select cast(saletime as date), salesid
from sales order by salesid limit 10;

saletime | salesid
-----+-----
2008-02-18 | 1
2008-06-06 | 2
2008-06-06 | 3
2008-06-09 | 4
2008-08-31 | 5
2008-07-16 | 6
2008-06-26 | 7
2008-07-10 | 8
2008-07-22 | 9
2008-08-06 | 10
(10 rows)
```

If you didn't use CAST as illustrated in the previous sample, the results would include the time:  
*2008-02-18 02:36:48.*

The following query casts variable character data to a date. It doesn't require sample data to run.

```
select cast('2008-02-18 02:36:48' as date) as mysaletime;

mysaletime

2008-02-18
(1 row)
```

In this example, the values in a date column are cast as timestamps:

```
select cast(caldate as timestamp), dateid
from date order by dateid limit 10;

caldate | dateid
-----+-----
2008-01-01 00:00:00 | 1827
2008-01-02 00:00:00 | 1828
2008-01-03 00:00:00 | 1829
2008-01-04 00:00:00 | 1830
2008-01-05 00:00:00 | 1831
2008-01-06 00:00:00 | 1832
2008-01-07 00:00:00 | 1833
2008-01-08 00:00:00 | 1834
2008-01-09 00:00:00 | 1835
2008-01-10 00:00:00 | 1836
(10 rows)
```

In a case like the previous sample, you can gain additional control over output formatting by using [TO\\_CHAR](#).

In this example, an integer is cast as a character string:

```
select cast(2008 as char(4));

bpchar

2008
```

In this example, a DECIMAL(6,3) value is cast as a DECIMAL(4,1) value:

```
select cast(109.652 as decimal(4,1));
numeric

109.7
```

This example shows a more complex expression. The PRICEPAID column (a DECIMAL(8,2) column) in the SALES table is converted to a DECIMAL(38,2) column and the values are multiplied by 10000000000000000000000000000000:

```
select salesid, pricepaid::decimal(38,2)*10000000000000000000000000000000
as value from sales where salesid<10 order by salesid;
```

| salesid | value                              |
|---------|------------------------------------|
| 1       | 7280000000000000000000000000000.00 |
| 2       | 7600000000000000000000000000000.00 |
| 3       | 3500000000000000000000000000000.00 |
| 4       | 1750000000000000000000000000000.00 |
| 5       | 1540000000000000000000000000000.00 |
| 6       | 3940000000000000000000000000000.00 |
| 7       | 7880000000000000000000000000000.00 |
| 8       | 1970000000000000000000000000000.00 |
| 9       | 5910000000000000000000000000000.00 |

(9 rows)

### Note

You can't perform a CAST or CONVERT operation on the GEOMETRY data type to change it to another data type. However, you can provide a hexadecimal representation of a string literal in extended well-known binary (EWKB) format as input to functions that accept a GEOMETRY argument. For example, the ST\_AsText function following expects a GEOMETRY data type.

```
SELECT ST_AsText('01010000000000000000000000000001C40000000000000002040');
```

```
st_astext

POINT(7 8)
```

You can also explicitly specify the GEOMETRY data type.

```
SELECT ST_AsText('01010000000000000000000000000001440000000000000000000000000000001840')::geometry;
```

```
st_astext

POINT(5 6)
```

## CONVERT function

Like the [CAST function](#), the CONVERT function converts one data type to another compatible data type. For instance, you can convert a string to a date, or a numeric type to a string. CONVERT performs a runtime conversion, which means that the conversion doesn't change a value's data type in a source table. It's changed only in the context of the query.

Certain data types require an explicit conversion to other data types using the CONVERT function. Other data types can be converted implicitly, as part of another command, without using CAST or CONVERT. See [Type compatibility and conversion \(p. 597\)](#).

## Syntax

```
CONVERT (type, expression)
```

## Arguments

*type*

One of the supported [Data types \(p. 574\)](#).

*expression*

An expression that evaluates to one or more values, such as a column name or a literal. Converting null values returns nulls. The expression cannot contain blank or empty strings.

## Return type

CONVERT returns the data type specified by the *type* argument.

### Note

Amazon Redshift returns an error if you try to perform a problematic conversion, such as a DECIMAL conversion that loses precision, like the following:

```
SELECT CONVERT(decimal(2,1), 123.456);
```

or an INTEGER conversion that causes an overflow:

```
SELECT CONVERT(smallint, 12345678);
```

## Examples

Some of the examples use the sample [TICKIT database](#). For more information about setting up sample data, see [Getting started with Amazon Redshift clusters and data loading](#).

The following query uses the CONVERT function to convert a column of decimals into integers

```
SELECT CONVERT(integer, pricepaid)
FROM sales WHERE salesid=100;
```

This example converts an integer into a character string.

```
SELECT CONVERT(char(4), 2008);
```

In this example, the current date and time is converted to a variable character data type:

```
SELECT CONVERT(VARCHAR(30), GETDATE());
getdate

2023-02-02 04:31:16
```

This example converts the saletime column into just the time, removing the dates from each row.

```
SELECT CONVERT(time, saletime), salesid
FROM sales order by salesid limit 10;
```

The following example converts variable character data into a datetime object.

```
SELECT CONVERT(datetime, '2008-02-18 02:36:48') as mysaletime;
```

## Note

You can't perform a CAST or CONVERT operation on the GEOMETRY data type to change it to another data type. However, you can provide a hexadecimal representation of a string literal in extended well-known binary (EWKB) format as input to functions that accept a GEOMETRY argument. For example, the ST\_AsText function following expects a GEOMETRY data type.

```
st_astext

POINT(7 8)
```

You can also explicitly specify the GEOMETRY data type.

```
st_astext

POINT(5 6)
```

## TO CHAR

`TO_CHAR` converts a timestamp or numeric expression to a character-string data format.

## Syntax

`TO CHAR (timestamp expression | numeric expression , 'format')`

## Arguments

*timestamp expression*

An expression that results in a `TIMESTAMP` or `TIMESTAMPTZ` type value or a value that can implicitly be coerced to a timestamp.

*numeric expression*

An expression that results in a numeric data type value or a value that can implicitly be coerced to a numeric type. For more information, see [Numeric types \(p. 575\)](#). TO\_CHAR inserts a space to the left of the numeral string.

## Note

TO CHAR does not support 128-bit DECIMAL values.

format

The format for the new value. For valid formats, see [Datetime format strings \(p. 1080\)](#) and [Numeric format strings \(p. 1082\)](#).

## Return type

### VARCHAR

## Examples

The following example converts a timestamp to a value with the date and time in a format with the name of the month padded to nine characters, the name of the day of the week, and the day number of the month.

```
select to_char(timestamp '2009-12-31 23:15:59', 'MONTH-DY-DD-YYYY HH12:MIPM');

to_char

DECEMBER -THU-31-2009 11:15PM
```

The following example converts a timestamp to a value with day number of the year.

```
select to_char(timestamp '2009-12-31 23:15:59', 'DDD');

to_char

365
```

The following example converts a timestamp to an ISO day number of the week.

```
select to_char(timestamp '2022-05-16 23:15:59', 'ID');

to_char

1
```

The following example extracts the month name from a date.

```
select to_char(date '2009-12-31', 'MONTH');

to_char

DECEMBER
```

The following example converts each STARTTIME value in the EVENT table to a string that consists of hours, minutes, and seconds.

```
select to_char(starttime, 'HH12:MI:SS')
from event where eventid between 1 and 5
order by eventid;

to_char

02:30:00
08:00:00
02:30:00
02:30:00
07:00:00
(5 rows)
```

The following example converts an entire timestamp value into a different format.

```
select starttime, to_char(starttime, 'MON-DD-YYYY HH12:MIPM')
from event where eventid=1;

starttime | to_char
-----+-----
```

```
2008-01-25 14:30:00 | JAN-25-2008 02:30PM
(1 row)
```

The following example converts a timestamp literal to a character string.

```
select to_char(timestamp '2009-12-31 23:15:59','HH24:MI:SS');
to_char

23:15:59
(1 row)
```

The following example converts a number to a character string with the negative sign at the end.

```
select to_char(-125.8, '999D99S');
to_char

125.80-
(1 row)
```

The following example converts a number to a character string with the currency symbol.

```
select to_char(-125.88, '$999D99');
to_char

$-125.88
(1 row)
```

The following example converts a number to a character string using angle brackets for negative numbers.

```
select to_char(-125.88, '$999D99PR');
to_char

$<-125.88>
(1 row)
```

The following example converts a number to a Roman numeral string.

```
select to_char(125, 'RN');
to_char

CXXV
(1 row)
```

The following example subtracts the commission from the price paid in the sales table. The difference is then rounded up and converted to a roman numeral, shown in the to\_char column:

```
select salesid, pricepaid, commission, (pricepaid - commission)
as difference, to_char(pricepaid - commission, 'rn') from sales
group by sales.pricepaid, sales.commission, salesid
order by salesid limit 10;

salesid | pricepaid | commission | difference | to_char
-----+-----+-----+-----+-----+
 1 | 728.00 | 109.20 | 618.80 | dcxix
 2 | 76.00 | 11.40 | 64.60 | lxv
 3 | 350.00 | 52.50 | 297.50 | ccxcvii
 4 | 175.00 | 26.25 | 148.75 | cxlix
```

|           |        |        |        |         |
|-----------|--------|--------|--------|---------|
| 5         | 154.00 | 23.10  | 130.90 | cxxxii  |
| 6         | 394.00 | 59.10  | 334.90 | cccxxxv |
| 7         | 788.00 | 118.20 | 669.80 | dclxx   |
| 8         | 197.00 | 29.55  | 167.45 | clxvii  |
| 9         | 591.00 | 88.65  | 502.35 | dii     |
| 10        | 65.00  | 9.75   | 55.25  | lv      |
| (10 rows) |        |        |        |         |

The following example adds the currency symbol to the difference values shown in the to\_char column:

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| select salesid, pricepaid, commission, (pricepaid - commission)<br>as difference, to_char(pricepaid - commission, '199999D99') from sales<br>group by sales.pricepaid, sales.commission, salesid<br>order by salesid limit 10;                                                                                                                                                                                                                                                                                                                  |
| salesid   pricepaid   commission   difference   to_char<br>-----+-----+-----+-----+-----<br>1   728.00   109.20   618.80   \$ 618.80<br>2   76.00   11.40   64.60   \$ 64.60<br>3   350.00   52.50   297.50   \$ 297.50<br>4   175.00   26.25   148.75   \$ 148.75<br>5   154.00   23.10   130.90   \$ 130.90<br>6   394.00   59.10   334.90   \$ 334.90<br>7   788.00   118.20   669.80   \$ 669.80<br>8   197.00   29.55   167.45   \$ 167.45<br>9   591.00   88.65   502.35   \$ 502.35<br>10   65.00   9.75   55.25   \$ 55.25<br>(10 rows) |

The following example lists the century in which each sale was made.

|                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| select salesid, saletime, to_char(saletime, 'cc') from sales<br>order by salesid limit 10;                                                                                                                                                                                                                                                                                                      |
| salesid   saletime   to_char<br>-----+-----+-----<br>1   2008-02-18 02:36:48   21<br>2   2008-06-06 05:00:16   21<br>3   2008-06-06 08:26:17   21<br>4   2008-06-09 08:38:52   21<br>5   2008-08-31 09:17:02   21<br>6   2008-07-16 11:59:24   21<br>7   2008-06-26 12:56:06   21<br>8   2008-07-10 02:12:36   21<br>9   2008-07-22 02:23:17   21<br>10   2008-08-06 02:51:55   21<br>(10 rows) |

The following example converts each STARTTIME value in the EVENT table to a string that consists of hours, minutes, seconds, and time zone.

|                                                                                                             |
|-------------------------------------------------------------------------------------------------------------|
| select to_char(starttime, 'HH12:MI:SS TZ')<br>from event where eventid between 1 and 5<br>order by eventid; |
| to_char<br>-----<br>02:30:00 UTC<br>08:00:00 UTC<br>02:30:00 UTC<br>02:30:00 UTC<br>07:00:00 UTC            |

```
(5 rows)
(10 rows)
```

The following example shows formatting for seconds, milliseconds, and microseconds.

```
select sysdate,
 to_char(sysdate, 'HH24:MI:SS') as seconds,
 to_char(sysdate, 'HH24:MI:SS.MS') as milliseconds,
 to_char(sysdate, 'HH24:MI:SS:US') as microseconds;
timestamp | seconds | milliseconds | microseconds
-----+-----+-----+-----
2015-04-10 18:45:09 | 18:45:09 | 18:45:09.325 | 18:45:09:325143
```

## TO\_DATE function

TO\_DATE converts a date represented by a character string to a DATE data type.

### Syntax

```
TO_DATE(string, format)
```

```
TO_DATE(string, format, is_strict)
```

### Arguments

*string*

A string to be converted.

*format*

A string literal that defines the format of the input *string*, in terms of its date parts. For a list of valid day, month, and year formats, see [Datetime format strings \(p. 1080\)](#).

*is\_strict*

An optional Boolean value that specifies whether an error is returned if an input date value is out of range. When *is\_strict* is set to TRUE, an error is returned if there is an out of range value. When *is\_strict* is set to FALSE, which is the default, then overflow values are accepted.

### Return type

TO\_DATE returns a DATE, depending on the *format* value.

If the conversion to *format* fails, then an error is returned.

### Examples

The following SQL statement converts the date 02 Oct 2001 into a date data type.

```
select to_date('02 Oct 2001', 'DD Mon YYYY');
to_date

2001-10-02
(1 row)
```

The following SQL statement converts the string 20010631 to a date.

```
select to_date('20010631', 'YYYYMMDD', FALSE);
```

The result is July 1, 2001, because there are only 30 days in June.

```
to_date

2001-07-01
```

The following SQL statement converts the string 20010631 to a date:

```
to_date('20010631', 'YYYYMMDD', TRUE);
```

The result is an error because there are only 30 days in June.

```
ERROR: date/time field date value out of range: 2001-6-31
```

## TO\_NUMBER

TO\_NUMBER converts a string to a numeric (decimal) value.

### Syntax

```
to_number(string, format)
```

### Arguments

*string*

String to be converted. The format must be a literal value.

*format*

The second argument is a format string that indicates how the character string should be parsed to create the numeric value. For example, the format '99D999' specifies that the string to be converted consists of five digits with the decimal point in the third position. For example, `to_number('12.345', '99D999')` returns 12.345 as a numeric value. For a list of valid formats, see [Numeric format strings \(p. 1082\)](#).

### Return type

TO\_NUMBER returns a DECIMAL number.

If the conversion to *format* fails, then an error is returned.

### Examples

The following example converts the string 12,454.8- to a number:

```
select to_number('12,454.8-', '99G999D9S');

to_number

-12454.8
```

The following example converts the string \$ 12,454.88 to a number:

```
select to_number('$ 12,454.88', 'L 99G999D99');
to_number

12454.88
```

The following example converts the string \$ 2,012,454.88 to a number:

```
select to_number('$ 2,012,454.88', 'L 9,999,999.99');
to_number

2012454.88
```

## TEXT\_TO\_INT\_ALT

TEXT\_TO\_INT\_ALT converts a character string to an integer using Teradata-style formatting. Fraction digits in the result are truncated.

### Syntax

```
TEXT_TO_INT_ALT (expression [, 'format')]
```

### Arguments

*expression*

An expression that results in one or more CHAR or VARCHAR values, such as a column name or literal string. Converting null values returns nulls. The function converts blank or empty strings to 0.

*format*

A string literal that defines the format of the input expression. For more information about the formatting characters you can specify, see [Teradata-style formatting characters for numeric data \(p. 1083\)](#).

### Return type

TEXT\_TO\_INT\_ALT returns an INTEGER value.

The fractional portion of the cast result is truncated.

Amazon Redshift returns an error if the conversion to the *format* phrase that you specify isn't successful.

### Examples

The following example converts the input *expression* string '123-' to the integer -123.

```
select text_to_int_alt('123-');
```

```
text_to_int_alt

-123
```

The following example converts the input *expression* string '2147483647+' to the integer 2147483647.

```
select text_to_int_alt('2147483647+');
```

```
text_to_int_alt

2147483647
```

The following example converts the exponential input *expression* string '-123E-2' to the integer -1.

```
select text_to_int_alt('-123E-2');
```

```
text_to_int_alt

-1
```

The following example converts the input *expression* string '2147483647+' to the integer 2147483647.

```
select text_to_int_alt('2147483647+');
```

```
text_to_int_alt

2147483647
```

The following example converts the input *expression* string '123{' with the *format* phrase '999S' to the integer 1230. The S character indicates a Signed Zoned Decimal. For more information, see [Teradata-style formatting characters for numeric data \(p. 1083\)](#).

```
text_to_int_alt('123{', '999S');
```

```
text_to_int_alt

1230
```

The following example converts the input *expression* string 'USD123' with the *format* phrase 'C9(I)' to the integer 123. See [Teradata-style formatting characters for numeric data \(p. 1083\)](#).

```
text_to_int_alt('USD123', 'C9(I)');
```

```
text_to_int_alt

123
```

The following example specifies a table column as the input *expression*.

```
select text_to_int_alt(a), text_to_int_alt(b) from t_text2int order by 1;
```

| text_to_int_alt | text_to_int_alt |
|-----------------|-----------------|
| -123            | -123            |
| -123            | -123            |
| 123             | 123             |
| 123             | 123             |

Following is the table definition and the insert statement for this example.

```
create table t_text2int (a varchar(200), b char(200));
```

```
insert into t_text2int VALUES('123', '123'),('123.123', '123.123'), ('-123', '-123'),
('123-', '123-');
```

## TEXT\_TO\_NUMERIC\_ALT

TEXT\_TO\_NUMERIC\_ALT performs a Teradata-style cast operation to convert a character string to a numeric data format.

### Syntax

```
TEXT_TO_NUMERIC_ALT (expression [, 'format'] [, precision, scale])
```

### Arguments

*expression*

An expression that evaluates to one or more CHAR or VARCHAR values, such as a column name or a literal. Converting null values returns nulls. Blank or empty strings are converted to 0.

*format*

A string literal that defines the format of the input expression. For more information, see [Teradata-style formatting characters for numeric data \(p. 1083\)](#).

*precision*

The number of digits in the numeric result. The default is 38.

*scale*

The number of digits to the right of the decimal point in the numeric result. The default is 0.

### Return type

TEXT\_TO\_NUMERIC\_ALT returns a DECIMAL number.

Amazon Redshift returns an error if the conversion to the *format* phrase that you specify isn't successful.

Amazon Redshift casts the input *expression* string to the numeric type with the highest precision that you specify for that type in the *precision* option. If the length of the numeric value exceeds the value that you specify for *precision*, Amazon Redshift rounds the numeric value according to the following rules:

- If the length of the cast result exceeds the length that you specify in the *format* phrase, Amazon Redshift returns an error.
- If the result is cast to a numeric value, the result is rounded to the closest value. If the fractional portion is exactly midway between the upper and lower cast result, the result is rounded to the nearest even value.

### Examples

The following example converts the input *expression* string '1.5' to the numeric value '2'. Because the statement doesn't specify *scale*, the *scale* defaults to 0 and the cast result doesn't include a fraction result. Because .5 is midway between 1 and 2, the cast result is rounded to the even value of 2.

```
select text_to_numeric_alt('1.5');
```

text\_to\_numeric\_alt

---

The following example converts the input *expression* string '2.51' to the numeric value 3. Because the statement doesn't specify a *scale* value, the *scale* defaults to 0 and the cast result doesn't include a fraction result. Because .51 is closer to 3 than 2, the cast result is rounded to the value of 3.

```
select text_to_numeric_alt('2.51');
```

text\_to\_numeric\_alt

---

The following example converts the input *expression* string 123.52501 with a *precision* of 10 and a *scale* of 2 to the numeric value 123.53.

```
select text_to_numeric_alt('123.52501', 10, 2);
```

text\_to\_numeric\_alt

---

The following example specifies a table column as the input *expression*.

```
select text_to_numeric_alt(a), text_to_numeric_alt(b) from t_text2numeric order by 1;
```

Following is the table definition and the insert statement for this example.

```
create table t_text2numeric (a varchar(200), b char(200));
```

```
insert into t_text2numeric values
('123', '123'),
('+123.456', '+123.456'),
('-' || repeat('9', 38), '-' || repeat('9', 38)),
(repeat('9', 38) || '+', repeat('9', 38) || '+'),
('-123E2', '-123E2');
```

# Datetime format strings

You can find a reference for datetime format strings following.

The following format strings apply to functions such as TO\_CHAR. These strings can contain datetime separators (such as '-' , '/' , or ':' ) and the following "dateparts" and "timeparts".

For examples of formatting dates as strings, see [TO\\_CHAR \(p. 1071\)](#).

| Datepart or timepart                           | Meaning                                                                                                                                                                                                                                                                                                                          |
|------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BC or B.C., AD or A.D., b.c. or bc, ad or a.d. | Upper and lowercase era indicators                                                                                                                                                                                                                                                                                               |
| CC                                             | Two-digit century number                                                                                                                                                                                                                                                                                                         |
| YYYY, YYYY, YY, Y                              | 4-digit, 3-digit, 2-digit, 1-digit year number                                                                                                                                                                                                                                                                                   |
| Y,YYY                                          | 4-digit year number with comma                                                                                                                                                                                                                                                                                                   |
| IYYY, IYY, IY, I                               | 4-digit, 3-digit, 2-digit, 1-digit International Organization for Standardization (ISO) year number                                                                                                                                                                                                                              |
| Q                                              | Quarter number (1 to 4)                                                                                                                                                                                                                                                                                                          |
| MONTH, Month, month                            | Month name (uppercase, mixed-case, lowercase, blank-padded to 9 characters)                                                                                                                                                                                                                                                      |
| MON, Mon, mon                                  | Abbreviated month name (uppercase, mixed-case, lowercase, blank-padded to 3 characters)                                                                                                                                                                                                                                          |
| MM                                             | Month number (01-12)                                                                                                                                                                                                                                                                                                             |
| RM, rm                                         | Month number in Roman numerals (I-XII, with I being January, uppercase or lowercase)                                                                                                                                                                                                                                             |
| W                                              | Week of month (1-5; the first week starts on the first day of the month.)                                                                                                                                                                                                                                                        |
| WW                                             | Week number of year (1-53; the first week starts on the first day of the year.)                                                                                                                                                                                                                                                  |
| IW                                             | ISO week number of year (the first Thursday of the new year is in week 1.)                                                                                                                                                                                                                                                       |
| DAY, Day, day                                  | Day name (uppercase, mixed-case, lowercase, blank-padded to 9 characters)                                                                                                                                                                                                                                                        |
| DY, Dy, dy                                     | Abbreviated day name (uppercase, mixed-case, lowercase, blank-padded to 3 characters)                                                                                                                                                                                                                                            |
| DDD                                            | Day of year (001-366)                                                                                                                                                                                                                                                                                                            |
| IDDD                                           | Day of ISO 8601 week-numbering year (001-371; day 1 of the year is Monday of the first ISO week)                                                                                                                                                                                                                                 |
| DD                                             | Day of month as a number (01-31)                                                                                                                                                                                                                                                                                                 |
| D                                              | Day of week (1-7; Sunday is 1)<br><b>Note</b><br>The D datepart behaves differently from the day of week (DOW) datepart used for the datetime functions DATE_PART and EXTRACT. DOW is based on integers 0-6, where Sunday is 0. For more information, see <a href="#">Date parts for date or timestamp functions (p. 1127)</a> . |

| Datepart or timepart                           | Meaning                                                            |
|------------------------------------------------|--------------------------------------------------------------------|
| ID                                             | ISO 8601 day of the week, Monday (1) to Sunday (7)                 |
| J                                              | Julian day (days since January 1, 4712 BC)                         |
| HH24                                           | Hour (24-hour clock, 00–23)                                        |
| HH or HH12                                     | Hour (12-hour clock, 01–12)                                        |
| MI                                             | Minutes (00–59)                                                    |
| SS                                             | Seconds (00–59)                                                    |
| MS                                             | Milliseconds (.000)                                                |
| US                                             | Microseconds (.000000)                                             |
| AM or PM, A.M. or P.M., a.m. or p.m., am or pm | Upper and lowercase meridian indicators (for 12-hour clock)        |
| TZ, tz                                         | Upper and lowercase time zone abbreviation; valid for TSTAMPZ only |
| OF                                             | Offset from UTC; valid for TSTAMPZ only                            |

**Note**

You must surround datetime separators (such as '-' , '/' or ':') with single quotation marks, but you must surround the "dateparts" and "timeparts" listed in the preceding table with double quotation marks.

## Numeric format strings

Following, you can find a reference for numeric format strings.

The following format strings apply to functions such as TO\_NUMBER and TO\_CHAR.

- For examples of formatting strings as numbers, see [TO\\_NUMBER \(p. 1076\)](#).
- For examples of formatting numbers as strings, see [TO\\_CHAR \(p. 1071\)](#).

| Format        | Description                                                                                     |
|---------------|-------------------------------------------------------------------------------------------------|
| 9             | Numeric value with the specified number of digits.                                              |
| 0             | Numeric value with leading zeros.                                                               |
| . (period), D | Decimal point.                                                                                  |
| , (comma)     | Thousands separator.                                                                            |
| CC            | Century code. For example, the 21st century started on 2001-01-01 (supported for TO_CHAR only). |
| FM            | Fill mode. Suppress padding blanks and zeroes.                                                  |
| PR            | Negative value in angle brackets.                                                               |

| Format   | Description                                                                                   |
|----------|-----------------------------------------------------------------------------------------------|
| S        | Sign anchored to a number.                                                                    |
| L        | Currency symbol in the specified position.                                                    |
| G        | Group separator.                                                                              |
| MI       | Minus sign in the specified position for numbers that are less than 0.                        |
| PL       | Plus sign in the specified position for numbers that are greater than 0.                      |
| SG       | Plus or minus sign in the specified position.                                                 |
| RN       | Roman numeral between 1 and 3999 (supported for TO_CHAR only).                                |
| TH or th | Ordinal number suffix. Does not convert fractional numbers or values that are less than zero. |

## Teradata-style formatting characters for numeric data

Following, you can find how the TEXT\_TO\_INT\_ALT and TEXT\_TO\_NUMERIC\_ALT functions interpret the characters in the input *expression* string. You can also find a list of the characters that you can specify in the *format* phrase. In addition, you can find a description of the differences between Teradata-style formatting and Amazon Redshift for the *format* option.

| Format  | Description                                                                                                                                                                                                                                                                                                                                                  |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| G       | Not supported as a group separator in the input <i>expression</i> string. You can't specify this character in the <i>format</i> phrase.                                                                                                                                                                                                                      |
| D       | Radix symbol. You can specify this character in the <i>format</i> phrase. This character is equivalent to the . (period).<br><br>The Radix symbol can't appear in a <i>format</i> phrase that contains any of the following characters: <ul style="list-style-type: none"><li>• . (period)</li><li>• S (uppercase 's')</li><li>• V (uppercase 'v')</li></ul> |
| / , : % | Insertion characters / (forward slash), comma (,), :, (colon), and % (percent sign).<br><br>You can't include these characters in the <i>format</i> phrase.<br><br>Amazon Redshift ignores these characters in the input <i>expression</i> string.                                                                                                           |
| .       | Period as a radix character, that is a decimal point.                                                                                                                                                                                                                                                                                                        |

| Format           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                  | <p>This character can't appear in a <i>format</i> phrase that contains any of the following characters:</p> <ul style="list-style-type: none"> <li>• D (uppercase 'd')</li> <li>• S (uppercase 's')</li> <li>• V (uppercase 'v')</li> </ul>                                                                                                                                                                                                                                                      |
| B                | You can't include the blank space character (B) in the <i>format</i> phrase. In the input <i>expression</i> string, leading and trailing spaces are ignored and spaces between digits aren't allowed.                                                                                                                                                                                                                                                                                            |
| + -              | You can't include the plus sign (+) or minus sign (-) in the <i>format</i> phrase. However, the plus sign (+) and minus sign (-) are parsed implicitly as part of numeric value if they appear in the input <i>expression</i> string.                                                                                                                                                                                                                                                            |
| V                | <p>Decimal point position indicator.</p> <p>This character can't appear in a <i>format</i> phrase that contains any of the following characters:</p> <ul style="list-style-type: none"> <li>• D (uppercase 'd')</li> <li>• . (period)</li> </ul>                                                                                                                                                                                                                                                 |
| Z                | Zero-suppressed decimal digit. Amazon Redshift trims leading zeros. The Z character can't follow a 9 character. The Z character must be to the left of the radix character if the fraction part contains the 9 character.                                                                                                                                                                                                                                                                        |
| 9                | Decimal digit.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| CHAR( <i>n</i> ) | <p>For this format, you can specify the following:</p> <ul style="list-style-type: none"> <li>• CHAR consists of Z or 9 characters. Amazon Redshift doesn't support a + (plus) or - (minus) in the CHAR value.</li> <li>• <i>n</i> is an integer constant, I, or F. For I, this is the number of characters necessary to display the integer portion of numeric or integer data. For F, this is the number of characters necessary to display the fractional portion of numeric data.</li> </ul> |
| -                | <p>Hyphen (-) character.</p> <p>You can't include this character in the <i>format</i> phrase.</p> <p>Amazon Redshift ignores this character in the input <i>expression</i> string.</p>                                                                                                                                                                                                                                                                                                           |

| Format              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| S                   | <p>Signed Zoned Decimal. The S character must follow the last decimal digit in the <i>format</i> phrase. The last character of the input <i>expression</i> string and the corresponding numeric conversion are listed in <a href="#">Data formatting characters for Signed Zone Decimal, Teradata-style numeric data formatting (p. 1086)</a>.</p> <p>The S character can't appear in a <i>format</i> phrase that contains any of the following characters:</p> <ul style="list-style-type: none"> <li>• + (plus sign)</li> <li>• . (period)</li> <li>• D (uppercase 'd')</li> <li>• Z (uppercase 'z')</li> <li>• F (uppercase 'f')</li> <li>• E (uppercase 'e')</li> </ul> |
| E                   | Exponential notation. The input <i>expression</i> string can include the exponent character. You can't specify E as an exponent character in <i>format</i> phrase.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| FN9                 | Not supported in Amazon Redshift.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| FNE                 | Not supported in Amazon Redshift.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| \$, USD, US Dollars | <p>Dollar sign (\$), ISO currency symbol (USD), and the currency name US Dollars.</p> <p>The ISO currency symbol USD and the currency name US Dollars are case-sensitive. Amazon Redshift supports only the USD currency. The input <i>expression</i> string can include spaces between the USD currency symbol and the numeric value, for example '\$ 123E2' or '123E2 \$'.</p>                                                                                                                                                                                                                                                                                            |
| L                   | Currency symbol. This currency symbol character can only appear once in the <i>format</i> phrase. You can't specify repeated currency symbol characters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| C                   | ISO currency symbol. This currency symbol character can only appear once in the <i>format</i> phrase. You can't specify repeated currency symbol characters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| N                   | Full currency name. This currency symbol character can only appear once in the <i>format</i> phrase. You can't specify repeated currency symbol characters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| O                   | Dual currency symbol. You can't specify this character in the <i>format</i> phrase.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| U                   | Dual ISO currency symbol. You can't specify this character in the <i>format</i> phrase.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

| Format | Description                                                                            |
|--------|----------------------------------------------------------------------------------------|
| A      | Full dual currency name. You can't specify this character in the <i>format</i> phrase. |

## Data formatting characters for Signed Zone Decimal, Teradata-style numeric data formatting

You can use the following characters in the *format* phrase of the TEXT\_TO\_INT\_ALT and TEXT\_TO\_NUMERIC\_ALT functions for a signed zoned decimal value.

| Last character of the input string | Numeric conversion |
|------------------------------------|--------------------|
| { or 0                             | <i>n</i> ... 0     |
| A or 1                             | <i>n</i> ... 1     |
| B or 2                             | <i>n</i> ... 2     |
| C or 3                             | <i>n</i> ... 3     |
| D or 4                             | <i>n</i> ... 4     |
| E or 5                             | <i>n</i> ... 5     |
| F or 6                             | <i>n</i> ... 6     |
| G or 7                             | <i>n</i> ... 7     |
| H or 8                             | <i>n</i> ... 8     |
| I or 9                             | <i>n</i> ... 9     |
| }                                  | - <i>n</i> ... 0   |
| J                                  | - <i>n</i> ... 1   |
| K                                  | - <i>n</i> ... 2   |
| L                                  | - <i>n</i> ... 3   |
| M                                  | - <i>n</i> ... 4   |
| N                                  | - <i>n</i> ... 5   |
| O                                  | - <i>n</i> ... 6   |
| P                                  | - <i>n</i> ... 7   |
| Q                                  | - <i>n</i> ... 8   |
| R                                  | - <i>n</i> ... 9   |

## Date and time functions

In this section, you can find information about the date and time scalar functions that Amazon Redshift supports.

## Topics

- [Summary of date and time functions \(p. 1087\)](#)
- [Date and time functions in transactions \(p. 1090\)](#)
- [Deprecated leader node-only functions \(p. 1091\)](#)
- [+ \(Concatenation\) operator \(p. 1091\)](#)
- [ADD\\_MONTHS function \(p. 1093\)](#)
- [AT TIME ZONE function \(p. 1094\)](#)
- [CONVERT\\_TIMEZONE function \(p. 1095\)](#)
- [CURRENT\\_DATE function \(p. 1097\)](#)
- [DATE\\_CMP function \(p. 1098\)](#)
- [DATE\\_CMP\\_TIMESTAMP function \(p. 1099\)](#)
- [DATE\\_CMP\\_TIMESTAMPTZ function \(p. 1100\)](#)
- [DATEADD function \(p. 1100\)](#)
- [DATEDIFF function \(p. 1103\)](#)
- [DATE\\_PART function \(p. 1106\)](#)
- [DATE\\_PART\\_YEAR function \(p. 1109\)](#)
- [DATE\\_TRUNC function \(p. 1110\)](#)
- [EXTRACT function \(p. 1112\)](#)
- [GETDATE function \(p. 1114\)](#)
- [INTERVAL\\_CMP function \(p. 1115\)](#)
- [LAST\\_DAY function \(p. 1116\)](#)
- [MONTHS\\_BETWEEN function \(p. 1117\)](#)
- [NEXT\\_DAY function \(p. 1118\)](#)
- [SYSDATE function \(p. 1119\)](#)
- [TIMEOFDAY function \(p. 1120\)](#)
- [TIMESTAMP\\_CMP function \(p. 1120\)](#)
- [TIMESTAMP\\_CMP\\_DATE function \(p. 1122\)](#)
- [TIMESTAMP\\_CMP\\_TIMESTAMPTZ function \(p. 1122\)](#)
- [TIMESTAMPTZ\\_CMP function \(p. 1123\)](#)
- [TIMESTAMPTZ\\_CMP\\_DATE function \(p. 1123\)](#)
- [TIMESTAMPTZ\\_CMP\\_TIMESTAMP function \(p. 1124\)](#)
- [TIMEZONE function \(p. 1124\)](#)
- [TO\\_TIMESTAMP function \(p. 1125\)](#)
- [TRUNC Date function \(p. 1126\)](#)
- [Date parts for date or timestamp functions \(p. 1127\)](#)

## Summary of date and time functions

| Function                                                                                                                                                         | Syntax             | Returns                  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|--------------------------|
| <a href="#">+ (Concatenation) operator (p. 1091)</a><br><br>Concatenates a date to a time on either side of the + symbol and returns a TIMESTAMP or TIMESTAMPTZ. | <i>date + time</i> | TIMESTAMP or TIMESTAMPTZ |

| Function                                                                                                                                                                                                             | Syntax                                                                                                               | Returns                           |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| <a href="#">ADD_MONTHS (p. 1093)</a><br><br>Adds the specified number of months to a date or timestamp.                                                                                                              | ADD_MONTHS<br>( <i>{date timestamp}</i> , <i>integer</i> )                                                           | TIMESTAMP                         |
| <a href="#">AT TIME ZONE (p. 1094)</a><br><br>Specifies which time zone to use with a TIMESTAMP or Timestamptz expression.                                                                                           | AT TIME ZONE ' <i>timezone</i> '                                                                                     | TIMESTAMP or<br>TIMESTAMPZ        |
| <a href="#">CONVERT_TIMEZONE (p. 1095)</a><br><br>Converts a timestamp from one time zone to another.                                                                                                                | CONVERT_TIMEZONE<br>([' <i>timezone</i> '], ' <i>timezone</i> ',<br><i>timestamp</i> )                               | TIMESTAMP                         |
| <a href="#">CURRENT_DATE (p. 1097)</a><br><br>Returns a date in the current session time zone (UTC by default) for the start of the current transaction.                                                             | CURRENT_DATE                                                                                                         | DATE                              |
| <a href="#">DATE_CMP (p. 1098)</a><br><br>Compares two dates and returns 0 if the dates are identical, 1 if <i>date1</i> is greater, and -1 if <i>date2</i> is greater.                                              | DATE_CMP ( <i>date1</i> , <i>date2</i> )                                                                             | INTEGER                           |
| <a href="#">DATE_CMP_TIMESTAMP (p. 1099)</a><br><br>Compares a date to a time and returns 0 if the values are identical, 1 if <i>date</i> is greater and -1 if <i>timestamp</i> is greater.                          | DATE_CMP_TIMESTAMP ( <i>date</i> ,<br><i>timestamp</i> )                                                             | INTEGER                           |
| <a href="#">DATE_CMP_TIMESTAMPTZ (p. 1100)</a><br><br>Compares a date and a timestamp with time zone and returns 0 if the values are identical, 1 if <i>date</i> is greater and -1 if <i>timestamptz</i> is greater. | DATE_CMP_TIMESTAMPTZ ( <i>date</i> ,<br><i>timestamptz</i> )                                                         | INTEGER                           |
| <a href="#">DATE_PART_YEAR (p. 1109)</a><br><br>Extracts the year from a date.                                                                                                                                       | DATE_PART_YEAR ( <i>date</i> )                                                                                       | INTEGER                           |
| <a href="#">DATEADD (p. 1100)</a><br><br>Increments a date or time by a specified interval.                                                                                                                          | DATEADD ( <i>datepart</i> , <i>interval</i> ,<br>( <i>{date time timetz timestamp}</i> ))                            | TIMESTAMP<br>or TIME or<br>TIMETZ |
| <a href="#">DATEDIFF (p. 1103)</a><br><br>Returns the difference between two dates or times for a given date part, such as a day or month.                                                                           | DATEDIFF ( <i>datepart</i> ,<br>( <i>{date time timetz timestamp}</i> ),<br>( <i>{date time timetz timestamp}</i> )) | BIGINT                            |
| <a href="#">DATE_PART (p. 1106)</a><br><br>Extracts a date part value from a date or time.                                                                                                                           | DATE_PART ( <i>datepart</i> ,<br>( <i>{date timestamp}</i> ))                                                        | DOUBLE                            |

| Function                                                                                                                                                                                          | Syntax                                    | Returns              |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|----------------------|
| <a href="#">DATE_TRUNC (p. 1110)</a><br><br>Truncates a timestamp based on a date part.                                                                                                           | DATE_TRUNC ('datepart',<br>timestamp)     | TIMESTAMP            |
| <a href="#">EXTRACT (p. 1112)</a><br><br>Extracts a date or time part from a timestamp, timestamptz, time, or timetz.                                                                             | EXTRACT (datepart FROM<br>source)         | INTEGER or<br>DOUBLE |
| <a href="#">GETDATE (p. 1114)</a><br><br>Returns the current date and time in the current session time zone (UTC by default). The parentheses are required.                                       | GETDATE()                                 | TIMESTAMP            |
| <a href="#">INTERVAL_CMP (p. 1115)</a><br><br>Compares two intervals and returns 0 if the intervals are equal, 1 if <i>interval1</i> is greater, and -1 if <i>interval2</i> is greater.           | INTERVAL_CMP (interval1,<br>interval2)    | INTEGER              |
| <a href="#">LAST_DAY (p. 1116)</a><br><br>Returns the date of the last day of the month that contains <i>date</i> .                                                                               | LAST_DAY(date)                            | DATE                 |
| <a href="#">MONTHS_BETWEEN (p. 1117)</a><br><br>Returns the number of months between two dates.                                                                                                   | MONTHS_BETWEEN (date, date)               | FLOAT8               |
| <a href="#">NEXT_DAY (p. 1118)</a><br><br>Returns the date of the first instance of <i>day</i> that is later than <i>date</i> .                                                                   | NEXT_DAY (date, day)                      | DATE                 |
| <a href="#">SYSDATE (p. 1119)</a><br><br>Returns the date and time in UTC for the start of the current transaction.                                                                               | SYSDATE                                   | TIMESTAMP            |
| <a href="#">TIMEOFDAY (p. 1120)</a><br><br>Returns the current weekday, date, and time in the current session time zone (UTC by default) as a string value.                                       | TIMEOFDAY()                               | VARCHAR              |
| <a href="#">TIMESTAMP_CMP (p. 1120)</a><br><br>Compares two timestamps and returns 0 if the timestamps are equal, 1 if <i>interval1</i> is greater, and -1 if <i>interval2</i> is greater.        | TIMESTAMP_CMP (timestamp1,<br>timestamp2) | INTEGER              |
| <a href="#">TIMESTAMP_CMP_DATE (p. 1122)</a><br><br>Compares a timestamp to a date and returns 0 if the values are identical, 1 if <i>timestamp</i> is greater, and -1 if <i>date</i> is greater. | TIMESTAMP_CMP_DATE<br>(timestamp, date)   | INTEGER              |

| Function                                                                                                                                                                                                                          | Syntax                                                                      | Returns                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|-----------------------------|
| <a href="#">TIMESTAMP_CMP_TIMESTAMPTZ (p. 1122)</a><br><br>Compares a timestamp with a timestamp with time zone and returns 0 if the values are equal, 1 if <i>timestamp</i> is greater, and -1 if <i>timestamptz</i> is greater. | TIMESTAMP_CMP_TIMESTAMPTZ<br>( <i>timestamp</i> , <i>timestamptz</i> )      | INTEGER                     |
| <a href="#">TIMESTAMPTZ_CMP (p. 1123)</a><br><br>Compares two timestamp with time zone values and returns 0 if the values are equal, 1 if <i>timestamptz1</i> is greater, and -1 if <i>timestamptz2</i> is greater.               | TIMESTAMPTZ_CMP<br>( <i>timestamptz1</i> , <i>timestamptz2</i> )            | INTEGER                     |
| <a href="#">TIMESTAMPTZ_CMP_DATE (p. 1123)</a><br><br>Compares the value of a timestamp with time zone and a date and returns 0 if the values are equal, 1 if <i>timestamptz</i> is greater, and -1 if <i>date</i> is greater.    | TIMESTAMPTZ_CMP_DATE<br>( <i>timestamptz</i> , <i>date</i> )                | INTEGER                     |
| <a href="#">TIMESTAMPTZ_CMP_TIMESTAMP (p. 1124)</a><br><br>Compares a timestamp with time zone with a timestamp and returns 0 if the values are equal, 1 if <i>timestamptz</i> is greater, and -1 if <i>timestamp</i> is greater. | TIMESTAMPTZ_CMP_TIMESTAMP<br>( <i>timestamptz</i> , <i>timestamp</i> )      | INTEGER                     |
| <a href="#">TIMEZONE (p. 1124)</a><br><br>Returns a timestamp for the specified time zone and timestamp value.                                                                                                                    | TIMEZONE (' <i>timezone</i> ' {<br><i>timestamp</i>   <i>timestamptz</i> }) | TIMESTAMP or<br>TIMESTAMPTZ |
| <a href="#">TO_TIMESTAMP (p. 1125)</a><br><br>Returns a timestamp with time zone for the specified timestamp and time zone format.                                                                                                | TO_TIMESTAMP (' <i>timestamp</i> ',<br>' <i>format</i> ')                   | TIMESTAMPTZ                 |
| <a href="#">TRUNC (p. 1126)</a><br><br>Truncates a timestamp and returns a date.                                                                                                                                                  | TRUNC( <i>timestamp</i> )                                                   | DATE                        |

#### Note

Leap seconds are not considered in elapsed-time calculations.

## Date and time functions in transactions

When you run the following functions within a transaction block (BEGIN ... END), the function returns the start date or time of the current transaction, not the start of the current statement.

- SYSDATE
- TIMESTAMP
- CURRENT\_DATE

The following functions always return the start date or time of the current statement, even when they are within a transaction block.

- GETDATE
- TIMEOFDAY

## Deprecated leader node-only functions

The following date functions are deprecated because they run only on the leader node. For more information, see [Leader node-only functions \(p. 1020\)](#).

- AGE. Use [DATEDIFF function \(p. 1103\)](#) instead.
- CURRENT\_TIME. Use [GETDATE function \(p. 1114\)](#) or [SYSDATE \(p. 1119\)](#) instead.
- CURRENT\_TIMESTAMP. Use [GETDATE function \(p. 1114\)](#) or [SYSDATE \(p. 1119\)](#) instead.
- LOCALTIME. Use [GETDATE function \(p. 1114\)](#) or [SYSDATE \(p. 1119\)](#) instead.
- LOCALTIMESTAMP. Use [GETDATE function \(p. 1114\)](#) or [SYSDATE \(p. 1119\)](#) instead.
- ISFINITE
- NOW. Use [GETDATE function \(p. 1114\)](#) or [SYSDATE \(p. 1119\)](#) instead.

## + (Concatenation) operator

Concatenates a DATE to a TIME or TIMETZ on either side of the + symbol and returns a TIMESTAMP or TIMESTAMPTZ.

### Syntax

*date* + *time*

*date* + *timetz*

The order of the arguments can be reversed.

### Arguments

*date*

A DATE column or an expression that implicitly converts to a DATE.

*time*

A TIME column or an expression that implicitly converts to a TIME.

*timetz*

A TIMETZ column or an expression that implicitly converts to a TIMETZ.

### Return type

TIMESTAMP if input is *time*.

TIMESTAMPTZ if input is *timetz*.

## Examples with a time column

The following example table TIME\_TEST has a column TIME\_VAL (type TIME) with three values inserted.

```
select time_val from time_test;

time_val

20:00:00
00:00:00.5550
00:58:00
```

The following example concatenates a date literal and a TIME\_VAL column.

```
select date '2000-01-02' + time_val as ts from time_test;

ts

2000-01-02 20:00:00
2000-01-02 00:00:00.5550
2000-01-02 00:58:00
```

The following example concatenates a date literal and a time literal.

```
select date '2000-01-01' + time '20:00:00' as ts;

ts

2000-01-01 20:00:00
```

## Examples with a TIMETZ column

The following example table TIMETZ\_TEST has a column TIMETZ\_VAL (type TIMETZ) with three values inserted.

```
select timetz_val from timetz_test;

timetz_val

04:00:00+00
00:00:00.5550+00
05:58:00+00
```

The following example concatenates a date literal and a TIMETZ\_VAL column.

```
select date '2000-01-01' + timetz_val as ts from timetz_test;
ts

2000-01-01 04:00:00+00
2000-01-01 00:00:00.5550+00
2000-01-01 05:58:00+00
```

The following example concatenates a DATE literal and a TIMETZ literal.

```
select date '2000-01-01' + timetz '20:00:00 PST' as ts;

ts
```

```

2000-01-02 04:00:00+00
```

## ADD\_MONTHS function

ADD\_MONTHS adds the specified number of months to a date or timestamp value or expression. The [DATEADD \(p. 1100\)](#) function provides similar functionality.

### Syntax

```
ADD_MONTHS(date | timestamp, integer)
```

### Arguments

*date | timestamp*

A date or timestamp column or an expression that implicitly converts to a date or timestamp. If the date is the last day of the month, or if the resulting month is shorter, the function returns the last day of the month in the result. For other dates, the result contains the same day number as the date expression.

*integer*

A positive or negative integer. Use a negative number to subtract months from dates.

### Return type

TIMESTAMP

### Example

The following query uses the ADD\_MONTHS function inside a TRUNC function. The TRUNC function removes the time of day from the result of ADD\_MONTHS. The ADD\_MONTHS function adds 12 months to each value from the CALDATE column.

```
select distinct trunc(add_months(caldate, 12)) as calplus12,
trunc(caldate) as cal
from date
order by 1 asc;

calplus12 | cal
-----+-----
2009-01-01 | 2008-01-01
2009-01-02 | 2008-01-02
2009-01-03 | 2008-01-03
...
(365 rows)
```

The following examples demonstrate the behavior when the ADD\_MONTHS function operates on dates with months that have different numbers of days.

```
select add_months('2008-03-31',1);

add_months

2008-04-30 00:00:00
(1 row)
```

```
select add_months('2008-04-30',1);

add_months

2008-05-31 00:00:00
(1 row)
```

## AT TIME ZONE function

AT TIME ZONE specifies which time zone to use with a TIMESTAMP or TIMESTAMPTZ expression.

### Syntax

```
AT TIME ZONE 'timezone'
```

### Arguments

*timezone*

The time zone for the return value. The time zone can be specified as a time zone name (such as '**Africa/Kampala**' or '**Singapore**') or as a time zone abbreviation (such as '**UTC**' or '**PDT**').

To view a list of supported time zone names, run the following command.

```
select pg_timezone_names();
```

To view a list of supported time zone abbreviations, run the following command.

```
select pg_timezone_abbrevs();
```

For more information and examples, see [Time zone usage notes \(p. 1095\)](#).

### Return type

TIMESTAMPTZ when used with a TIMESTAMP expression. TIMESTAMP when used with a TIMESTAMPTZ expression.

### Examples

The following example converts a timestamp value without time zone and interprets it as MST time (UTC +7 in POSIX), which is then converted to PST (UTC+8 in POSIX) for display.

```
SELECT TIMESTAMP '2001-02-16 20:38:40' AT TIME ZONE 'MST';

timestampz

'2001-02-16 19:38:40-08'
```

The following example takes an input timestamp with a time zone value where the specified time zone is EST (UTC+5 in POSIX) and converts it to MST (UTC+7 in POSIX).

```
SELECT TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40-05' AT TIME ZONE 'MST';
```

```
timestamp

'2001-02-16 18:38:40'
```

## CONVERT\_TIMEZONE function

CONVERT\_TIMEZONE converts a timestamp from one time zone to another. The function automatically adjusts for Daylight saving time.

### Syntax

```
CONVERT_TIMEZONE (['source_timezone',] 'target_timezone', 'timestamp')
```

### Arguments

*source\_timezone*

(Optional) The time zone of the current timestamp. The default is UTC. For more information, see [Time zone usage notes \(p. 1095\)](#).

*target\_timezone*

The time zone for the new timestamp. For more information, see [Time zone usage notes \(p. 1095\)](#).

*timestamp*

A timestamp column or an expression that implicitly converts to a timestamp.

### Return type

TIMESTAMP

### Time zone usage notes

Either *source\_timezone* or *target\_timezone* can be specified as a time zone name (such as 'Africa/Kampala' or 'Singapore') or as a time zone abbreviation (such as 'UTC' or 'PDT').

#### Note

The results of using a time zone name or a time zone abbreviation can be different due to local seasonal time, such as, Daylight saving time.

To view a list of supported time zone names, run the following command.

```
select pg_timezone_names();
```

To view a list of supported time zone abbreviations, run the following command.

```
select pg_timezone_abbrevs();
```

For more information, see [Appendix: Time zone names and abbreviations \(p. 1709\)](#).

### Using a time zone name

If you specify a time zone using a time zone name, CONVERT\_TIMEZONE automatically adjusts for Daylight saving time (DST), or any other local seasonal protocol, such as Summer Time, Standard Time,

or Winter Time, that is in force for that time zone during the date and time specified by '*timestamp*'. For example, 'Europe/London' represents UTC in the winter and adds one hour in the summer.

### Using a time zone abbreviation

Time zone abbreviations represent a fixed offset from UTC. If you specify a time zone using a time zone abbreviation, `CONVERT_TIMEZONE` uses the fixed offset from UTC and does not adjust for any local seasonal protocol.

### Using POSIX-style format

A POSIX-style time zone specification is in the form *STDoffset* or *STDoffsetDST*, where *STD* is a time zone abbreviation, *offset* is the numeric offset in hours west from UTC, and *DST* is an optional daylight-savings zone abbreviation. Daylight savings time is assumed to be one hour ahead of the given offset.

POSIX-style time zone formats use positive offsets west of Greenwich, in contrast to the ISO-8601 convention, which uses positive offsets east of Greenwich.

The following are examples of POSIX-style time zones:

- PST8
- PST8PDT
- EST5
- EST5EDT

#### Note

Amazon Redshift doesn't validate POSIX-style time zone specifications, so it is possible to set the time zone to an invalid value. For example, the following command doesn't return an error, even though it sets the time zone to an invalid value.

```
set timezone to 'xxx36';
```

## Examples

Many of the examples use the TICKIT sample data set. For more information, see [Sample database](#).

The following example converts the timestamp value from the default UTC time zone to PST.

```
select convert_timezone('PST', '2008-08-21 07:23:54');
convert_timezone

2008-08-20 23:23:54
```

The following example converts the timestamp value in the LISTTIME column from the default UTC time zone to PST. Though the timestamp is within the daylight time period, it's converted to standard time because the target time zone is specified as an abbreviation (PST).

```
select listtime, convert_timezone('PST', listtime) from listing
where listid = 16;
listtime | convert_timezone
-----+-----
2008-08-24 09:36:12 2008-08-24 01:36:12
```

The following example converts a timestamp LISTTIME column from the default UTC time zone to US/Pacific time zone. The target time zone uses a time zone name, and the timestamp is within the daylight time period, so the function returns the daylight time.

```
select listtime, convert_timezone('US/Pacific', listtime) from listing
where listid = 16;

listtime | convert_timezone
-----+-----
2008-08-24 09:36:12 | 2008-08-24 02:36:12
```

The following example converts a timestamp string from EST to PST:

```
select convert_timezone('EST', 'PST', '20080305 12:25:29');

convert_timezone

2008-03-05 09:25:29
```

The following example converts a timestamp to US Eastern Standard Time because the target time zone uses a time zone name (America/New\_York) and the timestamp is within the standard time period.

```
select convert_timezone('America/New_York', '2013-02-01 08:00:00');

convert_timezone

2013-02-01 03:00:00
(1 row)
```

The following example converts the timestamp to US Eastern Daylight Time because the target time zone uses a time zone name (America/New\_York) and the timestamp is within the daylight time period.

```
select convert_timezone('America/New_York', '2013-06-01 08:00:00');

convert_timezone

2013-06-01 04:00:00
(1 row)
```

The following example demonstrates the use of offsets.

```
SELECT CONVERT_TIMEZONE('GMT','NEWZONE +2','2014-05-17 12:00:00') as newzone_plus_2,
CONVERT_TIMEZONE('GMT','NEWZONE-2:15','2014-05-17 12:00:00') as newzone_minus_2_15,
CONVERT_TIMEZONE('GMT','America/Los_Angeles+2','2014-05-17 12:00:00') as la_plus_2,
CONVERT_TIMEZONE('GMT','GMT+2','2014-05-17 12:00:00') as gmt_plus_2;

newzone_plus_2 | newzone_minus_2_15 | la_plus_2 | gmt_plus_2
-----+-----+-----+-----+
2014-05-17 10:00:00 | 2014-05-17 14:15:00 | 2014-05-17 10:00:00 | 2014-05-17 10:00:00
(1 row)
```

## CURRENT\_DATE function

CURRENT\_DATE returns a date in the current session time zone (UTC by default) in the default format: YYYY-MM-DD.

### Note

CURRENT\_DATE returns the start date for the current transaction, not for the start of the current statement. Consider the scenario where you start a transaction containing multiple

statements on 10/01/08 23:59, and the statement containing CURRENT\_DATE runs at 10/02/08 00:00. CURRENT\_DATE returns 10/01/08, not 10/02/08.

## Syntax

```
CURRENT_DATE
```

## Return type

DATE

## Examples

The following example returns the current date (in the AWS Region where the function runs).

```
select current_date;
date

2008-10-01
```

The following example creates a table, inserts a row where the default of column todays\_date is CURRENT\_DATE, and then selects all the rows in the table.

```
CREATE TABLE insert_dates(
 label varchar(128) NOT NULL,
 todays_date DATE DEFAULT CURRENT_DATE);

INSERT INTO insert_dates(label)
VALUES('Date row inserted');

SELECT * FROM insert_dates;

label | todays_date
-----+-----
Date row inserted | 2023-05-10
```

## DATE\_CMP function

DATE\_CMP compares two dates. The function returns 0 if the dates are identical, 1 if *date1* is greater, and -1 if *date2* is greater.

## Syntax

```
DATE_CMP(date1, date2)
```

## Arguments

*date1*

A date column or an expression that implicitly converts to a date.

*date2*

A date column or an expression that implicitly converts to a date.

## Return type

INTEGER

## Example

The following query compares the CALDATE column to the date January 4, 2008 and returns whether the value in CALDATE is before (-1), equal to (0), or after (1) January 4, 2008:

```
select caldate, '2008-01-04',
date_cmp(caldate,'2008-01-04')
from date
order by dateid
limit 10;

caldate | ?column? | date_cmp
-----+-----+-----
2008-01-01 | 2008-01-04 | -1
2008-01-02 | 2008-01-04 | -1
2008-01-03 | 2008-01-04 | -1
2008-01-04 | 2008-01-04 | 0
2008-01-05 | 2008-01-04 | 1
2008-01-06 | 2008-01-04 | 1
2008-01-07 | 2008-01-04 | 1
2008-01-08 | 2008-01-04 | 1
2008-01-09 | 2008-01-04 | 1
2008-01-10 | 2008-01-04 | 1
(10 rows)
```

## DATE\_CMP\_TIMESTAMP function

Compares a date to a timestamp and returns 0 if the values are identical, 1 if *date* is greater alphabetically and -1 if *timestamp* is greater.

## Syntax

|                                                  |
|--------------------------------------------------|
| <code>DATE_CMP_TIMESTAMP(date, timestamp)</code> |
|--------------------------------------------------|

## Arguments

*date*

A date column or an expression that implicitly converts to a date.

*timestamp*

A timestamp column or an expression that implicitly converts to a timestamp.

## Return type

INTEGER

## Examples

The following example compares the date 2008-06-18 to LISTTIME. Listings made before this date return 1; listings made after this date return -1.

|                                                     |
|-----------------------------------------------------|
| <code>select listid, '2008-06-18', listtime,</code> |
|-----------------------------------------------------|

```
date_cmp_timestamp('2008-06-18', listtime)
from listing
order by 1, 2, 3, 4
limit 10;

listid | ?column? | listtime | date_cmp_timestamp
-----+-----+-----+-----+
 1 | 2008-06-18 | 2008-01-24 06:43:29 | 1
 2 | 2008-06-18 | 2008-03-05 12:25:29 | 1
 3 | 2008-06-18 | 2008-11-01 07:35:33 | -1
 4 | 2008-06-18 | 2008-05-24 01:18:37 | 1
 5 | 2008-06-18 | 2008-05-17 02:29:11 | 1
 6 | 2008-06-18 | 2008-08-15 02:08:13 | -1
 7 | 2008-06-18 | 2008-11-15 09:38:15 | -1
 8 | 2008-06-18 | 2008-11-09 05:07:30 | -1
 9 | 2008-06-18 | 2008-09-09 08:03:36 | -1
10 | 2008-06-18 | 2008-06-17 09:44:54 | 1
(10 rows)
```

## DATE\_CMP\_TIMESTAMP function

`DATE_CMP_TIMESTAMP` compares a date to a timestamp with time zone. If the date and timestamp values are identical, the function returns 0. If the date is greater alphabetically, the function returns 1. If the timestamp is greater, the function returns -1.

### Syntax

```
DATE_CMP_TIMESTAMP(date, timestamptz)
```

### Arguments

*date*

A DATE column or an expression that implicitly converts to a date.

*timestamptz*

A TIMESTAMPTZ column or an expression that implicitly converts to a timestamp with a time zone.

### Return type

INTEGER

## DATEADD function

Increments a DATE, TIME, TIMETZ, or TIMESTAMP value by a specified interval.

### Syntax

```
DATEADD(datepart, interval, {date|time|timetz|timestamp})
```

### Arguments

*datepart*

The date part (year, month, day, or hour, for example) that the function operates on. For more information, see [Date parts for date or timestamp functions \(p. 1127\)](#).

*interval*

An integer that specified the interval (number of days, for example) to add to the target expression.  
A negative integer subtracts the interval.

*date|time|timetz|timestamp*

A DATE, TIME, TIMETZ, or TIMESTAMP column or an expression that implicitly converts to a DATE, TIME, TIMETZ, or TIMESTAMP. The DATE, TIME, TIMETZ, or TIMESTAMP expression must contain the specified date part.

## Return type

TIMESTAMP or TIME or TIMETZ depending on the input data type.

### Examples with a DATE column

The following example adds 30 days to each date in November that exists in the DATE table.

```
select dateadd(day,30,caldate) as novplus30
from date
where month='NOV'
order by dateid;

novplus30

2008-12-01 00:00:00
2008-12-02 00:00:00
2008-12-03 00:00:00
...
(30 rows)
```

The following example adds 18 months to a literal date value.

```
select dateadd(month,18,'2008-02-28');

date_add

2009-08-28 00:00:00
(1 row)
```

The default column name for a DATEADD function is DATE\_ADD. The default timestamp for a date value is 00:00:00.

The following example adds 30 minutes to a date value that doesn't specify a timestamp.

```
select dateadd(m,30,'2008-02-28');

date_add

2008-02-28 00:30:00
(1 row)
```

You can name date parts in full or abbreviate them. In this case, *m* stands for minutes, not months.

### Examples with a TIME column

The following example table TIME\_TEST has a column TIME\_VAL (type TIME) with three values inserted.

```
select time_val from time_test;

time_val

20:00:00
00:00:00.5550
00:58:00
```

The following example adds 5 minutes to each TIME\_VAL in the TIME\_TEST table.

```
select dateadd(minute,5,time_val) as minplus5 from time_test;

minplus5

20:05:00
00:05:00.5550
01:03:00
```

The following example adds 8 hours to a literal time value.

```
select dateadd(hour, 8, time '13:24:55');

date_add

21:24:55
```

The following example shows when a time goes over 24:00:00 or under 00:00:00.

```
select dateadd(hour, 12, time '13:24:55');

date_add

01:24:55
```

## Examples with a TIMETZ column

The output values in these examples are in UTC which is the default timezone.

The following example table TIMETZ\_TEST has a column TIMETZ\_VAL (type TIMETZ) with three values inserted.

```
select timetz_val from timetz_test;

timetz_val

04:00:00+00
00:00:00.5550+00
05:58:00+00
```

The following example adds 5 minutes to each TIMETZ\_VAL in TIMETZ\_TEST table.

```
select dateadd(minute,5,timetz_val) as minplus5_tz from timetz_test;

minplus5_tz

04:05:00+00
00:05:00.5550+00
06:03:00+00
```

The following example adds 2 hours to a literal timetz value.

```
select dateadd(hour, 2, timetz '13:24:55 PST');

date_add

23:24:55+00
```

## Usage notes

The DATEADD(month, ...) and ADD\_MONTHS functions handle dates that fall at the ends of months differently:

- ADD\_MONTHS: If the date you are adding to is the last day of the month, the result is always the last day of the result month, regardless of the length of the month. For example, April 30 + 1 month is May 31.

```
select add_months('2008-04-30',1);

add_months

2008-05-31 00:00:00
(1 row)
```

- DATEADD: If there are fewer days in the date you are adding to than in the result month, the result is the corresponding day of the result month, not the last day of that month. For example, April 30 + 1 month is May 30.

```
select dateadd(month,1,'2008-04-30');

date_add

2008-05-30 00:00:00
(1 row)
```

The DATEADD function handles the leap year date 02-29 differently when using dateadd(month, 12,...) or dateadd(year, 1, ...).

```
select dateadd(month,12,'2016-02-29');

date_add

2017-02-28 00:00:00

select dateadd(year, 1, '2016-02-29');

date_add

2017-03-01 00:00:00
```

## DATEDIFF function

DATEDIFF returns the difference between the date parts of two date or time expressions.

### Syntax

```
DATEDIFF (datepart, {date|time|timetz|timestamp}, {date|time|timetz|timestamp})
```

## Arguments

### *datepart*

The specific part of the date or time value (year, month, or day, hour, minute, second, millisecond, or microsecond) that the function operates on. For more information, see [Date parts for date or timestamp functions \(p. 1127\)](#).

Specifically, DATEDIFF determines the number of date part boundaries that are crossed between two expressions. For example, suppose that you're calculating the difference in years between two dates, 12-31-2008 and 01-01-2009. In this case, the function returns 1 year despite the fact that these dates are only one day apart. If you are finding the difference in hours between two timestamps, 01-01-2009 8:30:00 and 01-01-2009 10:00:00, the result is 2 hours. If you are finding the difference in hours between two timestamps, 8:30:00 and 10:00:00, the result is 2 hours.

### *date|time|timetz|timestamp*

A DATE, TIME, TIMETZ, or TIMESTAMP column or expressions that implicitly convert to a DATE, TIME, TIMETZ, or TIMESTAMP. The expressions must both contain the specified date or time part. If the second date or time is later than the first date or time, the result is positive. If the second date or time is earlier than the first date or time, the result is negative.

## Return type

BIGINT

## Examples with a DATE column

The following example finds the difference, in number of weeks, between two literal date values.

```
select datediff(week, '2009-01-01', '2009-12-31') as numweeks;
numweeks

52
(1 row)
```

The following example finds the difference, in hours, between two literal date values. When you don't provide the time value for a date, it defaults to 00:00:00.

```
select datediff(hour, '2023-01-01', '2023-01-03 05:04:03');
date_diff

53
(1 row)
```

The following example finds the difference, in days, between two dates in the same row of a table.

```
select * from date_table;
start_date | end_date
-----+-----
2009-01-01 | 2009-03-23
2023-01-04 | 2024-05-04
(2 rows)

select datediff(day, start_date, end_date) as duration from date_table;
```

```
duration

 81
 486
(2 rows)
```

The following example finds the difference, in number of quarters, between a literal value in the past and today's date. This example assumes that the current date is June 5, 2008. You can name date parts in full or abbreviate them. The default column name for the DATEDIFF function is DATE\_DIFF.

```
select datediff(qtr, '1998-07-01', current_date);

date_diff

40
(1 row)
```

The following example joins the SALES and LISTING tables to calculate how many days after they were listed any tickets were sold for listings 1000 through 1005. The longest wait for sales of these listings was 15 days, and the shortest was less than one day (0 days).

```
select priceperticket,
 datediff(day, listtime, saletime) as wait
 from sales, listing
 where sales.listid = listing.listid
 and sales.listid between 1000 and 1005
 order by wait desc, priceperticket desc;

priceperticket | wait
-----+-----
 96.00 | 15
123.00 | 11
131.00 | 9
123.00 | 6
129.00 | 4
 96.00 | 4
 96.00 | 0
(7 rows)
```

This example calculates the average number of hours sellers waited for all ticket sales.

```
select avg(datediff(hours, listtime, saletime)) as avgwait
 from sales, listing
 where sales.listid = listing.listid;

avgwait

465
(1 row)
```

## Examples with a TIME column

The following example table TIME\_TEST has a column TIME\_VAL (type TIME) with three values inserted.

```
select time_val from time_test;

time_val

20:00:00
00:00:00.5550
```

00:58:00

The following example finds the difference in number of hours between the TIME\_VAL column and a time literal.

```
select datediff(hour, time_val, time '15:24:45') from time_test;

date_diff

-5
15
15
```

The following example finds the difference in number of minutes between two literal time values.

```
select datediff(minute, time '20:00:00', time '21:00:00') as nummins;

nummins

60
```

## Examples with a TIMETZ column

The following example table TIMETZ\_TEST has a column TIMETZ\_VAL (type TIMETZ) with three values inserted.

```
select timetz_val from timetz_test;

timetz_val

04:00:00+00
00:00:00.5550+00
05:58:00+00
```

The following example finds the differences in number of hours, between a TIMETZ literal and timetz\_val.

```
select datediff(hours, timetz '20:00:00 PST', timetz_val) as numhours from timetz_test;

numhours

0
-4
1
```

The following example finds the difference in number of hours, between two literal TIMETZ values.

```
select datediff(hours, timetz '20:00:00 PST', timetz '00:58:00 EST') as numhours;

numhours

1
```

## DATE\_PART function

DATE\_PART extracts date part values from an expression. DATE\_PART is a synonym of the PGDATE\_PART function.

## Syntax

```
DATE_PART(datepart, {date|timestamp})
```

## Arguments

*datepart*

An identifier literal or string of the specific part of the date value (for example, year, month, or day) that the function operates on. For more information, see [Date parts for date or timestamp functions \(p. 1127\)](#).

{*date|timestamp*}

A date column, timestamp column, or an expression that implicitly converts to a date or timestamp. The column or expression in *date* or *timestamp* must contain the date part specified in *datepart*.

## Return type

DOUBLE

## Examples

The default column name for the DATE\_PART function is pgdate\_part.

For more information about the data used in some of these examples, see [Sample database \(p. 1704\)](#).

The following example finds the minute from a timestamp literal.

```
SELECT DATE_PART(minute, timestamp '20230104 04:05:06.789');
pgdate_part

 5
```

The following example finds the week number from a timestamp literal. The week number calculation follows the ISO 8601 standard. For more information, see [ISO 8601](#) in Wikipedia.

```
SELECT DATE_PART(week, timestamp '20220502 04:05:06.789');
pgdate_part

 18
```

The following example finds the day of the month from a timestamp literal.

```
SELECT DATE_PART(day, timestamp '20220502 04:05:06.789');
pgdate_part

 2
```

The following example finds the day of the week from a timestamp literal. The week number calculation follows the ISO 8601 standard. For more information, see [ISO 8601](#) in Wikipedia.

```
SELECT DATE_PART(dayofweek, timestamp '20220502 04:05:06.789');
```

```
pgdate_part

 1
```

The following example finds the century from a timestamp literal. The century calculation follows the ISO 8601 standard. For more information, see [ISO 8601](#) in Wikipedia.

```
SELECT DATE_PART(century, timestamp '20220502 04:05:06.789');

pgdate_part

 21
```

The following example finds the millennium from a timestamp literal. The millennium calculation follows the ISO 8601 standard. For more information, see [ISO 8601](#) in Wikipedia.

```
SELECT DATE_PART(millennium, timestamp '20220502 04:05:06.789');

pgdate_part

 3
```

The following example finds the microseconds from a timestamp literal. The microseconds calculation follows the ISO 8601 standard. For more information, see [ISO 8601](#) in Wikipedia.

```
SELECT DATE_PART(microsecond, timestamp '20220502 04:05:06.789');

pgdate_part

 789000
```

The following example finds the month from a date literal.

```
SELECT DATE_PART(month, date '20220502');

pgdate_part

 5
```

The following example applies the DATE\_PART function to a column in a table.

```
SELECT date_part(w, listtime) AS weeks, listtime
FROM listing
WHERE listid=10

weeks | listtime
-----+-----
 25 | 2008-06-17 09:44:54
(1 row)
```

You can name date parts in full or abbreviate them; in this case, *w* stands for weeks.

The day of week date part returns an integer from 0-6, starting with Sunday. Use DATE\_PART with dow (DAYOFWEEK) to view events on a Saturday.

```
SELECT date_part(dow, starttime) AS dow, starttime
```

```
FROM event
WHERE date_part(dow, starttime)=6
ORDER BY 2,1;

dow | starttime
----+-----
6 | 2008-01-05 14:00:00
6 | 2008-01-05 14:00:00
6 | 2008-01-05 14:00:00
6 | 2008-01-05 14:00:00
...
(1147 rows)
```

## DATE\_PART\_YEAR function

The DATE\_PART\_YEAR function extracts the year from a date.

### Syntax

|                                   |
|-----------------------------------|
| <code>DATE_PART_YEAR(date)</code> |
|-----------------------------------|

### Argument

*date*

A date column or an expression that implicitly converts to a date.

### Return type

INTEGER

### Examples

The following example finds the year from a date literal.

|                                                                   |
|-------------------------------------------------------------------|
| <code>SELECT DATE_PART_YEAR(date '20220502 04:05:06.789');</code> |
| date_part_year                                                    |
| 2022                                                              |

The following example extracts the year from the CALDATE column. For more information about the data used in this example, see [Sample database \(p. 1704\)](#).

|                                                      |
|------------------------------------------------------|
| <code>select caldate, date_part_year(caldate)</code> |
| from date                                            |
| order by                                             |
| dateid limit 10;                                     |
| caldate   date_part_year                             |
| -----+-----                                          |
| 2008-01-01   2008                                    |
| 2008-01-02   2008                                    |
| 2008-01-03   2008                                    |
| 2008-01-04   2008                                    |
| 2008-01-05   2008                                    |
| 2008-01-06   2008                                    |
| 2008-01-07   2008                                    |
| 2008-01-08   2008                                    |
| 2008-01-09   2008                                    |

|                           |      |
|---------------------------|------|
| 2008-01-10  <br>(10 rows) | 2008 |
|---------------------------|------|

## DATE\_TRUNC function

The DATE\_TRUNC function truncates a timestamp expression or literal based on the date part that you specify, such as hour, day, or month.

### Syntax

```
DATE_TRUNC('datepart', timestamp)
```

### Arguments

*datepart*

The date part to which to truncate the timestamp value. The input *timestamp* is truncated to the precision of the input *datepart*. For example, month truncates to the first day of the month. Valid formats are as follows:

- microsecond, microseconds
- millisecond, milliseconds
- second, seconds
- minute, minutes
- hour, hours
- day, days
- week, weeks
- month, months
- quarter, quarters
- year, years
- decade, decades
- century, centuries
- millennium, millennia

For more information about abbreviations of some formats, see [Date parts for date or timestamp functions \(p. 1127\)](#)

*timestamp*

A timestamp column or an expression that implicitly converts to a timestamp.

### Return type

TIMESTAMP

### Examples

Truncate the input timestamp to the second.

```
SELECT DATE_TRUNC('second', TIMESTAMP '2020-04-30 04:05:06.789');
date_trunc
2020-04-30 04:05:06
```

Truncate the input timestamp to the minute.

```
SELECT DATE_TRUNC('minute', TIMESTAMP '2020-04-30 04:05:06.789');
date_trunc
2020-04-30 04:05:00
```

Truncate the input timestamp to the hour.

```
SELECT DATE_TRUNC('hour', TIMESTAMP '2020-04-30 04:05:06.789');
date_trunc
2020-04-30 04:00:00
```

Truncate the input timestamp to the day.

```
SELECT DATE_TRUNC('day', TIMESTAMP '2020-04-30 04:05:06.789');
date_trunc
2020-04-30 00:00:00
```

Truncate the input timestamp to the first day of a month.

```
SELECT DATE_TRUNC('month', TIMESTAMP '2020-04-30 04:05:06.789');
date_trunc
2020-04-01 00:00:00
```

Truncate the input timestamp to the first day of a quarter.

```
SELECT DATE_TRUNC('quarter', TIMESTAMP '2020-04-30 04:05:06.789');
date_trunc
2020-04-01 00:00:00
```

Truncate the input timestamp to the first day of a year.

```
SELECT DATE_TRUNC('year', TIMESTAMP '2020-04-30 04:05:06.789');
date_trunc
2020-01-01 00:00:00
```

Truncate the input timestamp to the first day of a century.

```
SELECT DATE_TRUNC('millennium', TIMESTAMP '2020-04-30 04:05:06.789');
date_trunc
2001-01-01 00:00:00
```

Truncate the input timestamp to the Monday of a week.

```
select date_trunc('week', TIMESTAMP '2022-04-30 04:05:06.789');
date_trunc
2022-04-25 00:00:00
```

In the following example, the DATE\_TRUNC function uses the 'week' date part to return the date for the Monday of each week.

```
select date_trunc('week', saletime), sum(pricepaid) from sales where
saletime like '2008-09%' group by date_trunc('week', saletime) order by 1;
```

| date_trunc | sum     |
|------------|---------|
| 2008-09-01 | 2474899 |
| 2008-09-08 | 2412354 |
| 2008-09-15 | 2364707 |
| 2008-09-22 | 2359351 |
| 2008-09-29 | 705249  |

## EXTRACT function

The EXTRACT function returns a date or time part from a TIMESTAMP, TIMESTAMPTZ, TIME, or TIMETZ value. Examples include a day, month, year, hour, minute, second, millisecond, or microsecond from a timestamp.

### Syntax

```
EXTRACT(datepart FROM source)
```

### Arguments

#### *datepart*

The subfield of a date or time to extract, such as a day, month, year, hour, minute, second, millisecond, or microsecond. For possible values, see [Date parts for date or timestamp functions \(p. 1127\)](#).

#### *source*

A column or expression that evaluates to a data type of TIMESTAMP, TIMESTAMPTZ, TIME, or TIMETZ.

### Return type

INTEGER if the *source* value evaluates to data type TIMESTAMP, TIME, or TIMETZ.

DOUBLE PRECISION if the *source* value evaluates to data type TIMESTAMPTZ.

### Examples with TIMESTAMP

The following example determines the week numbers for sales in which the price paid was \$10,000 or more. This example uses the TICKIT data. For more information, see [Sample database \(p. 1704\)](#)

| salesid | extract(week from saletime) as weeknum |
|---------|----------------------------------------|
| 159073  | 6                                      |
| 160318  | 8                                      |
| 161723  | 26                                     |

The following example returns the minute value from a literal timestamp value.

```
select extract(minute from timestamp '2009-09-09 12:08:43');
```

```
date_part

8
```

The following example returns the millisecond value from a literal timestamp value.

```
select extract(ms from timestamp '2009-09-09 12:08:43.101');

date_part

101
```

## Examples with TIMESTAMPTZ

The following example returns the year value from a literal timestamptz value.

```
select extract(year from timestamptz '1.12.1997 07:37:16.00 PST');

date_part

1997
```

## Examples with TIME

The following example table TIME\_TEST has a column TIME\_VAL (type TIME) with three values inserted.

```
select time_val from time_test;

time_val

20:00:00
00:00:00.5550
00:58:00
```

The following example extracts the minutes from each time\_val.

```
select extract(minute from time_val) as minutes from time_test;

minutes

0
0
58
```

The following example extracts the hours from each time\_val.

```
select extract(hour from time_val) as hours from time_test;

hours

20
0
0
```

The following example extracts milliseconds from a literal value.

```
select extract(ms from time '18:25:33.123456');
```

```
date_part

123
```

## Examples with TIMETZ

The following example table TIMETZ\_TEST has a column TIMETZ\_VAL (type TIMETZ) with three values inserted.

```
select timetz_val from timetz_test;
timetz_val

04:00:00+00
00:00:00.5550+00
05:58:00+00
```

The following example extracts the hours from each timetz\_val.

```
select extract(hour from timetz_val) as hours from time_test;
hours

4
0
5
```

The following example extracts milliseconds from a literal value. Literals aren't converted to UTC before the extraction is processed.

```
select extract(ms from timetz '18:25:33.123456 EST');
date_part

123
```

The following example returns the timezone offset hour from UTC from a literal timetz value.

```
select extract(timezone_hour from timetz '1.12.1997 07:37:16.00 PDT');
date_part

-7
```

## GETDATE function

GETDATE returns the current date and time in the current session time zone (UTC by default). It returns the start date or time of the current statement, even when it is within a transaction block.

### Syntax

```
GETDATE()
```

The parentheses are required.

## Return type

TIMESTAMP

## Examples

The following example uses the GETDATE function to return the full timestamp for the current date.

```
select getdate();
timestamp

2008-12-04 16:10:43
(1 row)
```

The following example uses the GETDATE function inside the TRUNC function to return the current date without the time.

```
select trunc(getdate());
trunc

2008-12-04
(1 row)
```

## INTERVAL\_CMP function

INTERVAL\_CMP compares two intervals and returns 1 if the first interval is greater, -1 if the second interval is greater, and 0 if the intervals are equal. For more information, see [Interval literals \(p. 590\)](#).

## Syntax

```
INTERVAL_CMP(interval1, interval2)
```

## Arguments

*interval1*

An interval literal value.

*interval2*

An interval literal value.

## Return type

INTEGER

## Examples

The following example compares the value of "3 days" to "1 year".

```
select interval_cmp('3 days', '1 year');
interval_cmp
```

```

-1
```

This example compares the value "7 days" to "1 week".

```
select interval_cmp('7 days','1 week');

interval_cmp

0
(1 row)
```

## LAST\_DAY function

`LAST_DAY` returns the date of the last day of the month that contains *date*. The return type is always DATE, regardless of the data type of the *date* argument.

For more information about retrieving specific date parts, see [DATE\\_TRUNC function \(p. 1110\)](#).

### Syntax

```
LAST_DAY ({ date | timestamp })
```

### Arguments

*date* | *timestamp*

A date or timestamp column or an expression that implicitly converts to a date or timestamp.

### Return type

DATE

### Examples

The following example returns the date of the last day in the current month.

```
select last_day(sysdate);

last_day

2014-01-31
(1 row)
```

The following example returns the number of tickets sold for each of the last 7 days of the month.

```
select datediff(day, saletime, last_day(saletime)) as "Days Remaining", sum(qtysold)
from sales
where datediff(day, saletime, last_day(saletime)) < 7
group by 1
order by 1;

days remaining | sum
-----+-----
0 | 10140
1 | 11187
```

|   |  |       |
|---|--|-------|
| 2 |  | 11515 |
| 3 |  | 11217 |
| 4 |  | 11446 |
| 5 |  | 11708 |
| 6 |  | 10988 |

(7 rows)

## MONTHS\_BETWEEN function

MONTHS\_BETWEEN determines the number of months between two dates.

If the first date is later than the second date, the result is positive; otherwise, the result is negative.

If either argument is null, the result is NULL.

### Syntax

```
MONTHS_BETWEEN (date1, date2)
```

### Arguments

*date1*

An expression, such as a column name, that evaluates to a valid date or timestamp value.

*date2*

An expression, such as a column name, that evaluates to a valid date or timestamp value.

### Return type

FLOAT8

The whole number portion of the result is based on the difference between the year and month values of the dates. The fractional portion of the result is calculated from the day and timestamp values of the dates and assumes a 31-day month.

If *date1* and *date2* both contain the same date within a month (for example, 1/15/14 and 2/15/14) or the last day of the month (for example, 8/31/14 and 9/30/14), then the result is a whole number based on the year and month values of the dates, regardless of whether the timestamp portion matches, if present.

### Examples

The following example returns the months between 1/18/1969 and 3/18/1969.

```
select months_between('1969-01-18', '1969-03-18')
as months;
months

-2
```

The following example returns the months between the first and last showings of an event.

```
select eventname,
```

```

min(starttime) as first_show,
max(starttime) as last_show,
months_between(max(starttime),min(starttime)) as month_diff
from event
group by eventname
order by eventname
limit 5;

eventname first_show last_show month_diff

.38 Special 2008-01-21 19:30:00.0 2008-12-25 15:00:00.0 11.12
3 Doors Down 2008-01-03 15:00:00.0 2008-12-01 19:30:00.0 10.94
70s Soul Jam 2008-01-16 19:30:00.0 2008-12-07 14:00:00.0 10.7
A Bronx Tale 2008-01-21 19:00:00.0 2008-12-15 15:00:00.0 10.8
A Catered Affair 2008-01-08 19:30:00.0 2008-12-19 19:00:00.0 11.35

```

## NEXT\_DAY function

`NEXT_DAY` returns the date of the first instance of the specified day that is later than the given date.

If the `day` value is the same day of the week as `given_date`, the next occurrence of that day is returned.

### Syntax

```
NEXT_DAY ({ date | timestamp }, day)
```

### Arguments

*date* | *timestamp*

A date or timestamp column or an expression that implicitly converts to a date or timestamp.

*day*

A string containing the name of any day. Capitalization does not matter.

Valid values are as follows.

| Day       | Values                   |
|-----------|--------------------------|
| Sunday    | Su, Sun, Sunday          |
| Monday    | M, Mo, Mon, Monday       |
| Tuesday   | Tu, Tue, Tues, Tuesday   |
| Wednesday | W, We, Wed, Wednesday    |
| Thursday  | Th, Thu, Thurs, Thursday |
| Friday    | F, Fr, Fri, Friday       |
| Saturday  | Sa, Sat, Saturday        |

### Return type

DATE

## Examples

The following example returns the date of the first Tuesday after 8/20/2014.

```
select next_day('2014-08-20', 'Tuesday');

next_day

2014-08-26
```

The following example gets target marketing dates for the third quarter.

```
select username, (firstname || ' ' || lastname) as name,
eventname, caldate, next_day(caldate, 'Monday') as marketing_target
from sales, date, users, event
where sales.buyerid = users.userid
and sales.eventid = event.eventid
and event.dateid = date.dateid
and date.qtr = 3
order by marketing_target, eventname, name;

username | name | eventname | caldate | marketing_target
-----+-----+-----+-----+-----+-----+
MB026QSG Callum Atkinson .38 Special 2008-07-06 2008-07-07
WCR50YIU Erasmus Alvarez A Doll's House 2008-07-03 2008-07-07
CKT700IE Hadassah Adkins Ana Gabriel 2008-07-06 2008-07-07
VVG070UO Nathan Abbott Armando Manzanero 2008-07-04 2008-07-07
GEW77SII Scarlet Avila August: Osage County 2008-07-06 2008-07-07
ECR71CVS Caryn Adkins Ben Folds 2008-07-03 2008-07-07
KUW82CYU Kaden Aguilar Bette Midler 2008-07-01 2008-07-07
WZE78DJZ Kay Avila Bette Midler 2008-07-01 2008-07-07
HXY04NVE Dante Austin Britney Spears 2008-07-02 2008-07-07
URY81YWF Wilma Anthony Britney Spears 2008-07-02 2008-07-07
```

## SYSDATE function

SYSDATE returns the current date and time in the current session time zone (UTC by default).

### Note

SYSDATE returns the start date and time for the current transaction, not for the start of the current statement.

## Syntax

```
SYSDATE
```

This function requires no arguments.

## Return type

TIMESTAMP

## Examples

The following example uses the SYSDATE function to return the full timestamp for the current date.

```
select sysdate;
timestamp
```

```

2008-12-04 16:10:43.976353
(1 row)
```

The following example uses the SYSDATE function inside the TRUNC function to return the current date without the time.

```
select trunc(sysdate);

trunc

2008-12-04
(1 row)
```

The following query returns sales information for dates that fall between the date when the query is issued and whatever date is 120 days earlier.

```
select salesid, pricepaid, trunc(saletime) as saletime, trunc(sysdate) as now
from sales
where saletime between trunc(sysdate)-120 and trunc(sysdate)
order by saletime asc;

salesid | pricepaid | saletime | now
-----+-----+-----+-----
91535 | 670.00 | 2008-08-07 | 2008-12-05
91635 | 365.00 | 2008-08-07 | 2008-12-05
91901 | 1002.00 | 2008-08-07 | 2008-12-05
...
```

## TIMEOFDAY function

TIMEOFDAY is a special alias used to return the weekday, date, and time as a string value. It returns the time of day string for the current statement, even when it is within a transaction block.

### Syntax

```
TIMEOFDAY()
```

### Return type

VARCHAR

### Examples

The following example returns the current date and time by using the TIMEOFDAY function.

```
select timeofday();
timeofday

Thu Sep 19 22:53:50.333525 2013 UTC
(1 row)
```

## TIMESTAMP\_CMP function

Compares the value of two timestamps and returns an integer. If the timestamps are identical, the function returns 0. If the first timestamp is greater alphabetically, the function returns 1. If the second timestamp is greater, the function returns -1.

## Syntax

```
TIMESTAMP_CMP(timestamp1, timestamp2)
```

## Arguments

*timestamp1*

A TIMESTAMP column or an expression that implicitly converts to a timestamp.

*timestamp2*

A TIMESTAMP column or an expression that implicitly converts to a timestamp.

## Return type

INTEGER

## Examples

The following example compares the LISTTIME and SALETIME for a listing. The value for TIMESTAMP\_CMP is -1 for all listings because the timestamp for the sale is after the timestamp for the listing.

```
select listing.listid, listing.listtime,
sales.saletime, timestamp_cmp(listing.listtime, sales.saletime)
from listing, sales
where listing.listid=sales.listid
order by 1, 2, 3, 4
limit 10;

listid | listtime | saletime | timestamp_cmp
-----+-----+-----+-----
 1 | 2008-01-24 06:43:29 | 2008-02-18 02:36:48 | -1
 4 | 2008-05-24 01:18:37 | 2008-06-06 05:00:16 | -1
 5 | 2008-05-17 02:29:11 | 2008-06-06 08:26:17 | -1
 5 | 2008-05-17 02:29:11 | 2008-06-09 08:38:52 | -1
 6 | 2008-08-15 02:08:13 | 2008-08-31 09:17:02 | -1
 10 | 2008-06-17 09:44:54 | 2008-06-26 12:56:06 | -1
 10 | 2008-06-17 09:44:54 | 2008-07-10 02:12:36 | -1
 10 | 2008-06-17 09:44:54 | 2008-07-16 11:59:24 | -1
 10 | 2008-06-17 09:44:54 | 2008-07-22 02:23:17 | -1
 12 | 2008-07-25 01:45:49 | 2008-08-04 03:06:36 | -1
(10 rows)
```

This example shows that TIMESTAMP\_CMP returns a 0 for identical timestamps:

```
select listid, timestamp_cmp(listtime, listtime)
from listing
order by 1 , 2
limit 10;

listid | timestamp_cmp
-----+-----
 1 | 0
 2 | 0
 3 | 0
 4 | 0
 5 | 0
 6 | 0
 7 | 0
```

|           |   |
|-----------|---|
| 8         | 0 |
| 9         | 0 |
| 10        | 0 |
| (10 rows) |   |

## TIMESTAMP\_CMP\_DATE function

`TIMESTAMP_CMP_DATE` compares the value of a timestamp and a date. If the timestamp and date values are identical, the function returns 0. If the timestamp is greater alphabetically, the function returns 1. If the date is greater, the function returns -1.

### Syntax

```
TIMESTAMP_CMP_DATE(timestamp, date)
```

### Arguments

*timestamp*

A timestamp column or an expression that implicitly converts to a timestamp.

*date*

A date column or an expression that implicitly converts to a date.

### Return type

INTEGER

### Examples

The following example compares LISTTIME to the date 2008-06-18. Listings made after this date return 1; listings made before this date return -1.

```
select listid, listtime,
 timestamp_cmp_date(listtime, '2008-06-18')
 from listing
 order by 1, 2, 3
 limit 10;

listid | listtime | timestamp_cmp_date
-----+-----+-----+
 1 | 2008-01-24 06:43:29 | -1
 2 | 2008-03-05 12:25:29 | -1
 3 | 2008-11-01 07:35:33 | 1
 4 | 2008-05-24 01:18:37 | -1
 5 | 2008-05-17 02:29:11 | -1
 6 | 2008-08-15 02:08:13 | 1
 7 | 2008-11-15 09:38:15 | 1
 8 | 2008-11-09 05:07:30 | 1
 9 | 2008-09-09 08:03:36 | 1
 10 | 2008-06-17 09:44:54 | -1
(10 rows)
```

## TIMESTAMP\_CMP\_TIMESTAMPTZ function

`TIMESTAMP_CMP_TIMESTAMPTZ` compares the value of a timestamp expression with a timestamp with time zone expression. If the timestamp and timestamp with time zone values are identical, the function

returns 0. If the timestamp is greater alphabetically, the function returns 1. If the timestamp with time zone is greater, the function returns -1.

## Syntax

```
TIMESTAMP_CMP_TIMESTAMPZ(timestamp, timestamptz)
```

### Arguments

*timestamp*

A TIMESTAMP column or an expression that implicitly converts to a timestamp.

*timestamptz*

A TIMESTAMP column or an expression that implicitly converts to a timestamp with a time zone.

### Return type

INTEGER

## TIMESTAMPTZ\_CMP function

TIMESTAMPTZ\_CMP compares the value of two timestamp with time zone values and returns an integer. If the timestamps are identical, the function returns 0. If the first timestamp is greater alphabetically, the function returns 1. If the second timestamp is greater, the function returns -1.

## Syntax

```
TIMESTAMPTZ_CMP(timestamptz1, timestamptz2)
```

### Arguments

*timestamptz1*

A TIMESTAMPTZ column or an expression that implicitly converts to a timestamp with time zone.

*timestamptz2*

A TIMESTAMPTZ column or an expression that implicitly converts to a timestamp with time zone.

### Return type

INTEGER

## TIMESTAMPTZ\_CMP\_DATE function

TIMESTAMPTZ\_CMP\_DATE compares the value of a timestamp and a date. If the timestamp and date values are identical, the function returns 0. If the timestamp is greater alphabetically, the function returns 1. If the date is greater, the function returns -1.

## Syntax

```
TIMESTAMPTZ_CMP_DATE(timestamptz, date)
```

## Arguments

*timestamptz*

A TIMESTAMPTZ column or an expression that implicitly converts to a timestamp with a time zone.  
*date*

A date column or an expression that implicitly converts to a date.

## Return type

INTEGER

## TIMESTAMPTZ\_CMP\_TIMESTAMP function

TIMESTAMPTZ\_CMP\_TIMESTAMP compares the value of a timestamp with time zone expression with a timestamp expression. If the timestamp with time zone and timestamp values are identical, the function returns 0. If the timestamp with time zone is greater alphabetically, the function returns 1. If the timestamp is greater, the function returns -1.

## Syntax

```
TIMESTAMPTZ_CMP_TIMESTAMP(timestamptz, timestamp)
```

## Arguments

*timestamptz*

A TIMESTAMPTZ column or an expression that implicitly converts to a timestamp with a time zone.  
*timestamp*

A TIMESTAMP column or an expression that implicitly converts to a timestamp.

## Return type

INTEGER

## TIMEZONE function

TIMEZONE returns a timestamp for the specified time zone and timestamp value.

For information and examples about how to set time zone, see [timezone \(p. 1700\)](#).

For information and examples about how to convert time zone, see [CONVERT\\_TIMEZONE \(p. 1095\)](#).

## Syntax

```
TIMEZONE ('timezone', { timestamp | timestamptz })
```

## Arguments

*timezone*

The time zone for the return value. The time zone can be specified as a time zone name (such as '**Africa/Kampala**' or '**Singapore**') or as a time zone abbreviation (such as '**UTC**' or '**PDT**'). To view a list of supported time zone names, run the following command.

```
select pg_timezone_names();
```

To view a list of supported time zone abbreviations, run the following command.

```
select pg_timezone_abbrevs();
```

For more information and examples, see [Time zone usage notes \(p. 1095\)](#).

#### *timestamp*

An expression that results in a TIMESTAMP type, or a value that can implicitly be coerced to a timestamp.

#### *timestamptz*

An expression that results in a Timestamptz type, or a value that can implicitly be coerced to a timestamp with time zone.

### Return type

Timestamptz when used with a TIMESTAMP expression.

Timestamp when used with a Timestamptz expression.

## TO\_TIMESTAMP function

TO\_TIMESTAMP converts a TIMESTAMP string to Timestamptz. For a list of additional date and time functions for Amazon Redshift, see [Date and time functions \(p. 1086\)](#).

### Syntax

```
to_timestamp (timestamp, format)
```

```
to_timestamp (timestamp, format, is_strict)
```

### Arguments

#### *timestamp*

A string that represents a timestamp value in the format specified by *format*. If this argument is left as empty, the timestamp value defaults to `0001-01-01 00:00:00`.

#### *format*

A string literal that defines the format of the *timestamp* value. Formats that include a time zone (`TZ`, `tz`, or `OF`) are not supported as input. For valid timestamp formats, see [Datetime format strings \(p. 1080\)](#).

#### *is\_strict*

An optional Boolean value that specifies whether an error is returned if an input timestamp value is out of range. When *is\_strict* is set to TRUE, an error is returned if there is an out of range value. When *is\_strict* is set to FALSE, which is the default, then overflow values are accepted.

### Return type

Timestamptz

## Examples

The following example demonstrates using the TO\_TIMESTAMP function to convert a TIMESTAMP string to a Timestamptz.

```
select sysdate, to_timestamp(sysdate, 'YYYY-MM-DD HH24:MI:SS') as second;

timestamp	second
2021-04-05 19:27:53.281812 | 2021-04-05 19:27:53+00
```

It's possible to pass TO\_TIMESTAMP part of a date. The remaining date parts are set to default values. The time is included in the output:

```
SELECT TO_TIMESTAMP('2017','YYYY');

to_timestamp

2017-01-01 00:00:00+00
```

The following SQL statement converts the string '2011-12-18 24:38:15' to a Timestamptz. The result is a Timestamptz that falls on the next day because the number of hours is more than 24 hours:

```
SELECT TO_TIMESTAMP('2011-12-18 24:38:15', 'YYYY-MM-DD HH24:MI:SS');

to_timestamp

2011-12-19 00:38:15+00
```

The following SQL statement converts the string '2011-12-18 24:38:15' to a Timestamptz. The result is an error because the time value in the timestamp is more than 24 hours:

```
SELECT TO_TIMESTAMP('2011-12-18 24:38:15', 'YYYY-MM-DD HH24:MI:SS', TRUE);

ERROR: date/time field time value out of range: 24:38:15.0
```

## TRUNC Date function

Truncates a timestamp and returns a date.

### Syntax

```
TRUNC(timestamp)
```

### Arguments

*timestamp*

A timestamp column or an expression that implicitly converts to a timestamp.

To return a timestamp value with `00:00:00` as the time, cast the function result to a `TIMESTAMP`.

### Return type

DATE

## Examples

The following example returns the date portion from the result of the SYSDATE function (which returns a timestamp).

```
select sysdate;
timestamp

2011-07-21 10:32:38.248109
(1 row)

select trunc(sysdate);

trunc

2011-07-21
(1 row)
```

The following example applies the TRUNC function to a TIMESTAMP column. The return type is a date.

```
select trunc(starttime) from event
order by eventid limit 1;

trunc

2008-01-25
(1 row)
```

## Date parts for date or timestamp functions

The following table identifies the date part and time part names and abbreviations that are accepted as arguments to the following functions:

- DATEADD
- DATEDIFF
- DATE\_PART
- EXTRACT

| Date part or time part | Abbreviations                                               |
|------------------------|-------------------------------------------------------------|
| millennium, millennia  | mil, mils                                                   |
| century, centuries     | c, cent, cents                                              |
| decade, decades        | dec, decs                                                   |
| epoch                  | epoch (supported by the <a href="#">EXTRACT (p. 1112)</a> ) |
| year, years            | y, yr, yrs                                                  |
| quarter, quarters      | qtr, qtrs                                                   |
| month, months          | mon, mons                                                   |
| week, weeks            | w                                                           |

| Date part or time part                   | Abbreviations                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| day of week                              | dayofweek, dow, dw, weekday (supported by the <a href="#">DATE_PART (p. 1106)</a> and the <a href="#">EXTRACT function (p. 1112)</a> )<br><br>Returns an integer from 0–6, starting with Sunday.<br><br><b>Note</b><br>The DOW date part behaves differently from the day of week (D) date part used for datetime format strings. D is based on integers 1–7, where Sunday is 1. For more information, see <a href="#">Datetime format strings (p. 1080)</a> . |
| day of year                              | dayofyear, doy, dy, yearday (supported by the <a href="#">EXTRACT (p. 1112)</a> )                                                                                                                                                                                                                                                                                                                                                                              |
| day, days                                | d                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| hour, hours                              | h, hr, hrs                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| minute, minutes                          | m, min, mins                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| second, seconds                          | s, sec, secs                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| millisecond, milliseconds                | ms, msec, msecs, msec, mseconds, millisec, millisecs, millisecon                                                                                                                                                                                                                                                                                                                                                                                               |
| microsecond, microseconds                | microsec, microsecs, microsecond, usecond, useconds, us, usec, usecs                                                                                                                                                                                                                                                                                                                                                                                           |
| timezone, timezone_hour, timezone_minute | Supported by the <a href="#">EXTRACT (p. 1112)</a> for timestamp with time zone (TIMESTAMPTZ) only.                                                                                                                                                                                                                                                                                                                                                            |

## Variations in results with seconds, milliseconds, and microseconds

Minor differences in query results occur when different date functions specify seconds, milliseconds, or microseconds as date parts:

- The EXTRACT function return integers for the specified date part only, ignoring higher- and lower-level date parts. If the specified date part is seconds, milliseconds and microseconds are not included in the result. If the specified date part is milliseconds, seconds and microseconds are not included. If the specified date part is microseconds, seconds and milliseconds are not included.
- The DATE\_PART function returns the complete seconds portion of the timestamp, regardless of the specified date part, returning either a decimal value or an integer as required.

For example, compare the results of the following queries:

```

create table seconds(micro timestamp);

insert into seconds values('2009-09-21 11:10:03.189717');

select extract(sec from micro) from seconds;
date_part

3
(1 row)

select date_part(sec, micro) from seconds;
pgdate_part

3.189717
(1 row)

```

## CENTURY, EPOCH, DECADE, and MIL notes

### CENTURY or CENTURIES

Amazon Redshift interprets a CENTURY to start with year ###1 and end with year ###0:

```
select extract (century from timestamp '2000-12-16 12:21:13');
date_part

20
(1 row)

select extract (century from timestamp '2001-12-16 12:21:13');
date_part

21
(1 row)
```

### EPOCH

The Amazon Redshift implementation of EPOCH is relative to 1970-01-01 00:00:00.000000 independent of the time zone where the cluster resides. You might need to offset the results by the difference in hours depending on the time zone where the cluster is located.

The following example demonstrates the following:

1. Creates a table called EVENT\_EXAMPLE based on the EVENT table. This CREATE AS command uses the DATE\_PART function to create a date column (called PGDATE\_PART by default) to store the epoch value for each event.
2. Selects the column and data type of EVENT\_EXAMPLE from PG\_TABLE\_DEF.
3. Selects EVENTNAME, STARTTIME, and PGDATE\_PART from the EVENT\_EXAMPLE table to view the different date and time formats.
4. Selects EVENTNAME and STARTTIME from EVENT EXAMPLE as is. Converts epoch values in PGDATE\_PART using a 1 second interval to a timestamp without time zone, and returns the results in a column called CONVERTED\_TIMESTAMP.

```
create table event_example
as select eventname, starttime, date_part(epoch, starttime) from event;

select "column", type from pg_table_def where tablename='event_example';

 column | type
-----+-----
eventname | character varying(200)
starttime | timestamp without time zone
pgdate_part | double precision
(3 rows)
```

```
select eventname, starttime, pgdate_part from event_example;

 eventname | starttime | pgdate_part
-----+-----+-----+
Mamma Mia! | 2008-01-01 20:00:00 | 1199217600
Spring Awakening | 2008-01-01 15:00:00 | 1199199600
Nas | 2008-01-01 14:30:00 | 1199197800
Hannah Montana | 2008-01-01 19:30:00 | 1199215800
K.D. Lang | 2008-01-01 15:00:00 | 1199199600
Spamalot | 2008-01-02 20:00:00 | 1199304000
Macbeth | 2008-01-02 15:00:00 | 1199286000
The Cherry Orchard | 2008-01-02 14:30:00 | 1199284200
```

|             |                                  |
|-------------|----------------------------------|
| Macbeth     | 2008-01-02 19:30:00   1199302200 |
| Demi Lovato | 2008-01-02 19:30:00   1199302200 |

```
select eventname,
starttime,
timestamp with time zone 'epoch' + pgdate_part * interval '1 second' AS
converted_timestamp
from event_example;
```

| eventname          | starttime           | converted_timestamp |
|--------------------|---------------------|---------------------|
| Mamma Mia!         | 2008-01-01 20:00:00 | 2008-01-01 20:00:00 |
| Spring Awakening   | 2008-01-01 15:00:00 | 2008-01-01 15:00:00 |
| Nas                | 2008-01-01 14:30:00 | 2008-01-01 14:30:00 |
| Hannah Montana     | 2008-01-01 19:30:00 | 2008-01-01 19:30:00 |
| K.D. Lang          | 2008-01-01 15:00:00 | 2008-01-01 15:00:00 |
| Spamalot           | 2008-01-02 20:00:00 | 2008-01-02 20:00:00 |
| Macbeth            | 2008-01-02 15:00:00 | 2008-01-02 15:00:00 |
| The Cherry Orchard | 2008-01-02 14:30:00 | 2008-01-02 14:30:00 |
| Macbeth            | 2008-01-02 19:30:00 | 2008-01-02 19:30:00 |
| Demi Lovato        | 2008-01-02 19:30:00 | 2008-01-02 19:30:00 |
| ...                |                     |                     |

## DECade or DECADES

Amazon Redshift interprets the DECADE or DECADES DATEPART based on the common calendar. For example, because the common calendar starts from the year 1, the first decade (decade 1) is 0001-01-01 through 0009-12-31, and the second decade (decade 2) is 0010-01-01 through 0019-12-31. For example, decade 201 spans from 2000-01-01 to 2009-12-31:

```
select extract(decade from timestamp '1999-02-16 20:38:40');
date_part

200
(1 row)

select extract(decade from timestamp '2000-02-16 20:38:40');
date_part

201
(1 row)

select extract(decade from timestamp '2010-02-16 20:38:40');
date_part

202
(1 row)
```

## MIL or MILS

Amazon Redshift interprets a MIL to start with the first day of year #001 and end with the last day of year #000:

```
select extract (mil from timestamp '2000-12-16 12:21:13');
date_part

2
(1 row)

select extract (mil from timestamp '2001-12-16 12:21:13');
date_part

```

|              |
|--------------|
| 3<br>(1 row) |
|--------------|

## Hash functions

### Topics

- [CHECKSUM function \(p. 1131\)](#)
- [farmFingerprint64 function \(p. 1132\)](#)
- [FUNC\\_SHA1 function \(p. 1132\)](#)
- [FNV\\_HASH function \(p. 1133\)](#)
- [MD5 function \(p. 1134\)](#)
- [SHA function \(p. 1135\)](#)
- [SHA1 function \(p. 1135\)](#)
- [SHA2 function \(p. 1135\)](#)
- [MURMUR3\\_32\\_HASH \(p. 1136\)](#)

A hash function is a mathematical function that converts a numerical input value into another value.

## CHECKSUM function

Computes a checksum value for building a hash index.

### Syntax

|                                          |
|------------------------------------------|
| <code>CHECKSUM(<i>expression</i>)</code> |
|------------------------------------------|

### Argument

*expression*

The input expression must be a VARCHAR, INTEGER, or DECIMAL data type.

### Return type

The CHECKSUM function returns an integer.

### Example

The following example computes a checksum value for the COMMISSION column:

|                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------|
| <pre>select checksum(commission) from sales order by salesid limit 10;  checksum ----- 10920 1140 5250 2625</pre> |
|-------------------------------------------------------------------------------------------------------------------|

```
2310
5910
11820
2955
8865
975
(10 rows)
```

## farmFingerprint64 function

Computes the farmhash value of the input argument using the `Fingerprint64` function.

### Syntax

```
farmFingerprint64(expression)
```

### Argument

*expression*

The input expression must be a VARCHAR or VARBYTE data type.

### Return type

The `farmFingerprint64` function returns a BIGINT.

### Example

The following example returns the `farmFingerprint64` value of Amazon Redshift that is input as a VARCHAR data type.

```
SELECT farmFingerprint64('Amazon Redshift');
```

```
farmfingerprint64

8085098817162212970
```

The following example returns the `farmFingerprint64` value of Amazon Redshift that is input as a VARBYTE data type.

```
SELECT farmFingerprint64('Amazon Redshift'::varbyte);
```

```
farmfingerprint64

8085098817162212970
```

## FUNC\_SHA1 function

Synonym of SHA1 function.

See [SHA1 function \(p. 1135\)](#).

## FNV\_HASH function

Computes the 64-bit FNV-1a non-cryptographic hash function for all basic data types.

### Syntax

```
FNV_HASH(value [, seed])
```

### Arguments

#### *value*

The input value to be hashed. Amazon Redshift uses the binary representation of the value to hash the input value; for instance, INTEGER values are hashed using 4 bytes and BIGINT values are hashed using 8 bytes. Also, hashing CHAR and VARCHAR inputs does not ignore trailing spaces.

#### *seed*

The BIGINT seed of the hash function is optional. If not given, Amazon Redshift uses the default FNV seed. This enables combining the hash of multiple columns without any conversions or concatenations.

### Return type

BIGINT

### Example

The following examples return the FNV hash of a number, the string 'Amazon Redshift', and the concatenation of the two.

```
select fnv_hash(1);
 fnv_hash

-5968735742475085980
(1 row)
```

```
select fnv_hash('Amazon Redshift');
 fnv_hash

7783490368944507294
(1 row)
```

```
select fnv_hash('Amazon Redshift', fnv_hash(1));
 fnv_hash

-2202602717770968555
(1 row)
```

### Usage notes

- To compute the hash of a table with multiple columns, you can compute the FNV hash of the first column and pass it as a seed to the hash of the second column. Then, it passes the FNV hash of the second column as a seed to the hash of the third column.

The following example creates seeds to hash a table with multiple columns.

```
select fnv_hash(column_3, fnv_hash(column_2, fnv_hash(column_1))) from sample_table;
```

- The same property can be used to compute the hash of a concatenation of strings.

```
select fnv_hash('abcd');
 fnv_hash

-281581062704388899
(1 row)
```

```
select fnv_hash('cd', fnv_hash('ab'));
 fnv_hash

-281581062704388899
(1 row)
```

- The hash function uses the type of the input to determine the number of bytes to hash. Use casting to enforce a specific type, if necessary.

The following examples use different types of input to produce different results.

```
select fnv_hash(1::smallint);
 fnv_hash

 589727492704079044
(1 row)
```

```
select fnv_hash(1);
 fnv_hash

-5968735742475085980
(1 row)
```

```
select fnv_hash(1::bigint);
 fnv_hash

-8517097267634966620
(1 row)
```

## MD5 function

Uses the MD5 cryptographic hash function to convert a variable-length string into a 32-character string that is a text representation of the hexadecimal value of a 128-bit checksum.

### Syntax

```
MD5(string)
```

### Arguments

*string*

A variable-length string.

## Return type

The MD5 function returns a 32-character string that is a text representation of the hexadecimal value of a 128-bit checksum.

## Examples

The following example shows the 128-bit value for the string 'Amazon Redshift':

```
select md5('Amazon Redshift');
md5

f7415e33f972c03abd4f3fed36748f7a
(1 row)
```

## SHA function

Synonym of SHA1 function.

See [SHA1 function \(p. 1135\)](#).

## SHA1 function

The SHA1 function uses the SHA1 cryptographic hash function to convert a variable-length string into a 40-character string that is a text representation of the hexadecimal value of a 160-bit checksum.

### Syntax

SHA1 is a synonym of [SHA function \(p. 1135\)](#) and [FUNC\\_SHA1 function \(p. 1132\)](#).

```
SHA1(string)
```

### Arguments

*string*

A variable-length string.

## Return type

The SHA1 function returns a 40-character string that is a text representation of the hexadecimal value of a 160-bit checksum.

## Example

The following example returns the 160-bit value for the word 'Amazon Redshift':

```
select sha1('Amazon Redshift');
```

## SHA2 function

The SHA2 function uses the SHA2 cryptographic hash function to convert a variable-length string into a character string. The character string is a text representation of the hexadecimal value of the checksum with the specified number of bits.

## Syntax

```
SHA2(string, bits)
```

## Arguments

*string*

A variable-length string.

*integer*

The number of bits in the hash functions. Valid values are 0 (same as 256), 224, 256, 384, and 512.

## Return type

The SHA2 function returns a character string that is a text representation of the hexadecimal value of the checksum or an empty string if the number of bits is invalid.

## Example

The following example returns the 256-bit value for the word 'Amazon Redshift':

```
select sha2('Amazon Redshift', 256);
```

## MURMUR3\_32\_HASH

The MURMUR3\_32\_HASH function computes the 32-bit Murmur3A non-cryptographic hash for all common data types including numeric and string types.

## Syntax

```
MURMUR3_32_HASH(value [, seed])
```

## Arguments

*value*

The input value to hash. Amazon Redshift hashes the binary representation of the input value. This behavior is similar to [FNV\\_HASH function \(p. 1133\)](#), but the value is converted to the binary representation specified by the [Apache Iceberg 32-bit Murmur3 hash specification](#).

*seed*

The INT seed of the hash function. This argument is optional. If not given, Amazon Redshift uses the default seed of 0. This enables combining the hash of multiple columns without any conversions or concatenations.

## Return type

The function returns an INT.

## Example

The following examples return the Murmur3 hash of a number, the string 'Amazon Redshift', and the concatenation of the two.

```
select MURMUR3_32_HASH(1);

 MURMUR3_32_HASH

-5968735742475085980
(1 row)
```

```
select MURMUR3_32_HASH('Amazon Redshift');

 MURMUR3_32_HASH

7783490368944507294
(1 row)
```

```
select MURMUR3_32_HASH('Amazon Redshift', MURMUR3_32_HASH(1));

 MURMUR3_32_HASH

-2202602717770968555
(1 row)
```

## Usage notes

To compute the hash of a table with multiple columns, you can compute the Murmur3 hash of the first column and pass it as a seed to the hash of the second column. Then, it passes the Murmur3 hash of the second column as a seed to the hash of the third column.

The following example creates seeds to hash a table with multiple columns.

```
select MURMUR3_32_HASH(column_3, MURMUR3_32_HASH(column_2, MURMUR3_32_HASH(column_1))) from
sample_table;
```

The same property can be used to compute the hash of a concatenation of strings.

```
select MURMUR3_32_HASH('abcd');

 MURMUR3_32_HASH

-281581062704388899
(1 row)
```

```
select MURMUR3_32_HASH('cd', MURMUR3_32_HASH('ab'));

 MURMUR3_32_HASH

-281581062704388899
(1 row)
```

The hash function uses the type of the input to determine the number of bytes to hash. Use casting to enforce a specific type, if necessary.

The following examples use different input types to produce different results.

```
select MURMUR3_32_HASH(1::smallint);

 MURMUR3_32_HASH

```

```
589727492704079044
(1 row)
```

```
select MURMUR3_32_HASH(1);

MURMUR3_32_HASH

-5968735742475085980
(1 row)
```

```
select MURMUR3_32_HASH(1::bigint);

MURMUR3_32_HASH

-8517097267634966620
(1 row)
```

## HyperLogLog functions

Following, you can find descriptions for the HyperLogLog functions for SQL that Amazon Redshift supports.

### Topics

- [HLL function \(p. 1138\)](#)
- [HLL\\_CREATE\\_SKETCH function \(p. 1139\)](#)
- [HLL\\_CARDINALITY function \(p. 1139\)](#)
- [HLL\\_COMBINE function \(p. 1140\)](#)
- [HLL\\_COMBINE\\_SKETCHES function \(p. 1141\)](#)

## HLL function

The HLL function returns the HyperLogLog cardinality of the input expression values. The HLL function works with any data types except the HLLSKETCH data type. The HLL function ignores NULL values. When there are no rows in a table or all rows are NULL, the resulting cardinality is 0.

### Syntax

```
HLL (aggregate_expression)
```

### Argument

*aggregate\_expression*

Any valid expression that provides the value to an aggregate, such as a column name. This function supports any data type as input except HLLSKETCH, GEOMETRY, GEOGRAPHY, and VARBYTE.

### Return type

The HLL function returns a BIGINT or INT8 value.

### Examples

The following example returns the cardinality of column an\_int in table a\_table.

```
CREATE TABLE a_table(an_int INT);
INSERT INTO a_table VALUES (1), (2), (3), (4);

SELECT hll(an_int) AS cardinality FROM a_table;
cardinality

4
```

## HLL\_CREATE\_SKETCH function

The HLL\_CREATE\_SKETCH function returns an HLLSKETCH data type that encapsulates the input expression values. The HLL\_CREATE\_SKETCH function works with any data type and ignores NULL values. When there are no rows in a table or all rows are NULL, the resulting sketch has no index-value pairs such as {"version":1,"logm":15,"sparse":{"indices":[],"values":[]}}.

### Syntax

```
HLL_CREATE_SKETCH (aggregate_expression)
```

### Argument

*aggregate\_expression*

Any valid expression that provides the value to an aggregate, such as a column name. NULL values are ignored. This function supports any data type as input except HLLSKETCH, GEOMETRY, GEOGRAPHY, and VARBYTE.

### Return type

The HLL\_CREATE\_SKETCH function returns an HLLSKETCH value.

### Examples

The following example returns the HLLSKETCH type for column an\_int in table a\_table. A JSON object is used to represent a sparse HyperLogLog sketch when importing, exporting, or printing sketches. A string representation (in Base64 format) is used to represent a dense HyperLogLog sketch.

```
CREATE TABLE a_table(an_int INT);
INSERT INTO a_table VALUES (1), (2), (3), (4);

SELECT hll_create_sketch(an_int) AS sketch FROM a_table;
sketch

>{"version":1,"logm":15,"sparse":{"indices":[20812342,20850007,22362299,47158030],"values":[1,2,1,1]}
(1 row)
```

## HLL\_CARDINALITY function

The HLL\_CARDINALITY function returns the cardinality of the input HLLSKETCH data type.

### Syntax

```
HLL_CARDINALITY (hllsketch_expression)
```

## Argument

*hllsketch\_expression*

Any valid expression that evaluates to an HLLSKETCH type, such as a column name. The input value is the HLLSKETCH data type.

## Return type

The HLL\_CARDINALITY function returns a BIGINT or INT8 value.

## Examples

The following example returns the cardinality of column sketch in table hll\_table.

```
CREATE TABLE a_table(an_int INT, b_int INT);
INSERT INTO a_table VALUES (1,1), (2,1), (3,1), (4,1), (1,2), (2,2), (3,2), (4,2), (5,2),
(6,2);

CREATE TABLE hll_table (sketch HLLSKETCH);
INSERT INTO hll_table select hll_create_sketch(an_int) from a_table group by b_int;

SELECT hll_cardinality(sketch) AS cardinality FROM hll_table;
cardinality

6
4
(2 rows)
```

## HLL\_COMBINE function

The HLL\_COMBINE aggregate function returns an HLLSKETCH data type that combines all input HLLSKETCH values.

The combination of two or more HyperLogLog sketches is a new HLLSKETCH that encapsulates information about the union of the distinct values that each input sketch represents. After combining sketches, Amazon Redshift extracts the cardinality of the union of two or more datasets. For more information on how to combine multiple sketches, see [Example: Return a HyperLogLog sketch from combining multiple sketches \(p. 307\)](#).

## Syntax

```
HLL_COMBINE (hllsketch_expression)
```

## Argument

*hllsketch\_expression*

Any valid expression that evaluates to an HLLSKETCH type, such as a column name. The input value is the HLLSKETCH data type.

## Return type

The HLL\_COMBINE function returns an HLLSKETCH type.

## Examples

The following example returns the combined HLLSKETCH values in the table hll\_table.

```

CREATE TABLE a_table(an_int INT, b_int INT);
INSERT INTO a_table VALUES (1,1), (2,1), (3,1), (4,1), (1,2), (2,2), (3,2), (4,2), (5,2),
(6,2);

CREATE TABLE hll_table (sketch HLLSKETCH);
INSERT INTO hll_table select hll_create_sketch(an_int) from a_table group by b_int;

SELECT hll_combine(sketch) AS sketches FROM hll_table;
sketches

>{"version":1,"logm":15,"sparse":{"indices":
[20812342,20850007,22362299,40314817,42650774,47158030],"values":[1,2,1,3,2,1]}
(1 row)

```

## HLL\_COMBINE\_SKETCHES function

The HLL\_COMBINE\_SKETCHES is a scalar function that takes as input two HLLSKETCH values and combines them into a single HLLSKETCH.

The combination of two or more HyperLogLog sketches is a new HLLSKETCH that encapsulates information about the union of the distinct values that each input sketch represents.

### Syntax

```
HLL_COMBINE_SKETCHES (hllsketch_expression1, hllsketch_expression2)
```

### Argument

*hllsketch\_expression1* and *hllsketch\_expression2*

Any valid expression that evaluates to an HLLSKETCH type, such as a column name.

### Return type

The HLL\_COMBINE\_SKETCHES function returns an HLLSKETCH type.

## Examples

The following example returns the combined HLLSKETCH values in the table hll\_table.

```

WITH tbl1(x, y)
AS (SELECT Hll_create_sketch(1),
 Hll_create_sketch(2)
 UNION ALL
 SELECT Hll_create_sketch(3),
 Hll_create_sketch(4)
 UNION ALL
 SELECT Hll_create_sketch(5),
 Hll_create_sketch(6)
 UNION ALL
 SELECT Hll_create_sketch(7),
 Hll_create_sketch(8)),
tbl2(x, y)
AS (SELECT Hll_create_sketch(9),

```

```
Hll_create_sketch(10)
UNION ALL
SELECT Hll_create_sketch(11),
 Hll_create_sketch(12)
UNION ALL
SELECT Hll_create_sketch(13),
 Hll_create_sketch(14)
UNION ALL
SELECT Hll_create_sketch(15),
 Hll_create_sketch(16)
UNION ALL
SELECT Hll_create_sketch(NULL),
 Hll_create_sketch(NULL)),
tbl3(x, y)
AS (SELECT *
 FROM tbl1
 UNION ALL
 SELECT *
 FROM tbl2)
SELECT Hll_combine_sketches(x, y)
FROM tbl3;
```

## JSON functions

### Topics

- [IS\\_VALID\\_JSON function \(p. 1143\)](#)
- [IS\\_VALID\\_JSON\\_ARRAY function \(p. 1144\)](#)
- [JSON\\_ARRAY\\_LENGTH function \(p. 1145\)](#)
- [JSON\\_EXTRACT\\_ARRAY\\_ELEMENT\\_TEXT function \(p. 1146\)](#)
- [JSON\\_EXTRACT\\_PATH\\_TEXT function \(p. 1147\)](#)
- [JSON\\_PARSE function \(p. 1149\)](#)
- [CAN\\_JSON\\_PARSE function \(p. 1150\)](#)
- [JSON\\_SERIALIZE function \(p. 1150\)](#)
- [JSON\\_SERIALIZE\\_TO\\_VARBYTE function \(p. 1151\)](#)

When you need to store a relatively small set of key-value pairs, you might save space by storing the data in JSON format. Because JSON strings can be stored in a single column, using JSON might be more efficient than storing your data in tabular format. For example, suppose you have a sparse table, where you need to have many columns to fully represent all possible attributes, but most of the column values are NULL for any given row or any given column. By using JSON for storage, you might be able to store the data for a row in key:value pairs in a single JSON string and eliminate the sparsely-populated table columns.

In addition, you can easily modify JSON strings to store additional key:value pairs without needing to add columns to a table.

We recommend using JSON sparingly. JSON isn't a good choice for storing larger datasets because, by storing disparate data in a single column, JSON doesn't use the Amazon Redshift column store architecture. Though Amazon Redshift supports JSON functions over CHAR and VARCHAR columns, we recommend using SUPER for processing data in JSON serialization format. SUPER uses a post-parse schemaless representation that can efficiently query hierarchical data. For more information about the SUPER data type, see [Ingesting and querying semistructured data in Amazon Redshift \(p. 368\)](#).

JSON uses UTF-8 encoded text strings, so JSON strings can be stored as CHAR or VARCHAR data types. Use VARCHAR if the strings include multi-byte characters.

JSON strings must be properly formatted JSON, according to the following rules:

- The root level JSON can either be a JSON object or a JSON array. A JSON object is an unordered set of comma-separated key:value pairs enclosed by curly braces.  
For example, `{"one":1, "two":2}`
- A JSON array is an ordered set of comma-separated values enclosed by brackets.  
An example is the following: `["first", {"one":1}, "second", 3, null]`
- JSON arrays use a zero-based index; the first element in an array is at position 0. In a JSON key:value pair, the key is a string in double quotation marks.
- A JSON value can be any of the following:
  - JSON object
  - JSON array
  - string in double quotation marks
  - number (integer and float)
  - boolean
  - null
- Empty objects and empty arrays are valid JSON values.
- JSON fields are case-sensitive.
- White space between JSON structural elements (such as `{ }`, `[ ]`) is ignored.

The Amazon Redshift JSON functions and the Amazon Redshift COPY command use the same methods to work with JSON-formatted data. For more information about working with JSON, see [COPY from JSON format \(p. 719\)](#)

## IS\_VALID\_JSON function

The IS\_VALID\_JSON function validates a JSON string. The function returns Boolean `true` (`t`) if the string is properly formed JSON or `false` (`f`) if the string is malformed. To validate a JSON array, use [IS\\_VALID\\_JSON\\_ARRAY function \(p. 1144\)](#)

For more information, see [JSON functions \(p. 1142\)](#).

### Syntax

```
is_valid_json('json_string')
```

### Arguments

*json\_string*

A string or expression that evaluates to a JSON string.

### Return type

BOOLEAN

### Example

The following example creates a table and inserts JSON strings for testing.

```
create table test_json(id int identity(0,1), json_strings varchar);
```

```
-- Insert valid JSON strings --
insert into test_json(json_strings) values
('{"a":2}'),
('{"a":{"b":{"c":1}}}''),
('{"a": [1,2,"b"]}'');

-- Insert invalid JSON strings --
insert into test_json(json_strings)values
('{}'),
('1:"a"'),
('[1,2,3]');
```

The following example validates the strings in the preceding example.

| id | json_strings          | is_valid_json |
|----|-----------------------|---------------|
| 0  | {"a":2}               | true          |
| 2  | {"a":{"b":{"c":1}}}'' | true          |
| 4  | {"a": [1,2,"b"]}'     | true          |
| 6  | {}                    | false         |
| 8  | 1:"a"                 | false         |
| 10 | [1,2,3]               | false         |

## IS\_VALID\_JSON\_ARRAY function

The IS\_VALID\_JSON\_ARRAY function validates a JSON array. The function returns Boolean true (t) if the array is properly formed JSON or false (f) if the array is malformed. To validate a JSON string, use [IS\\_VALID\\_JSON function \(p. 1143\)](#)

For more information, see [JSON functions \(p. 1142\)](#).

### Syntax

```
is_valid_json_array('json_array')
```

### Arguments

*json\_array*

A string or expression that evaluates to a JSON array.

### Return type

BOOLEAN

### Example

The following example creates a table and inserts JSON strings for testing.

```
create table test_json_arrays(id int identity(0,1), json_arrays varchar);

-- Insert valid JSON array strings --
insert into test_json_arrays(json_arrays)
values('[]'),
```

```
(['a","b"]'),
(['a',[{"b":1,[{"c":2,3,null}]}]);

-- Insert invalid JSON array strings --
insert into test_json_arrays(json_arrays) values
('{"a":1}'),
('a'),
('[1,2,]');
```

The following example validates the strings in the preceding example.

| json_arrays                   | is_valid_json_array |
|-------------------------------|---------------------|
| []                            | true                |
| ["a","b"]                     | true                |
| ["a",["b",1,[{"c":2,3,null}]] | true                |
| {"a":1}                       | false               |
| a                             | false               |
| [1,2,]                        | false               |

## JSON\_ARRAY\_LENGTH function

The `JSON_ARRAY_LENGTH` function returns the number of elements in the outer array of a JSON string. If the `null_if_invalid` argument is set to `true` and the JSON string is invalid, the function returns `NULL` instead of returning an error.

For more information, see [JSON functions \(p. 1142\)](#).

### Syntax

```
json_array_length('json_array' [, null_if_invalid])
```

### Arguments

*json\_array*

A properly formatted JSON array.

*null\_if\_invalid*

A Boolean value that specifies whether to return `NULL` if the input JSON string is invalid instead of returning an error. To return `NULL` if the JSON is invalid, specify `true` (`t`). To return an error if the JSON is invalid, specify `false` (`f`). The default is `false`.

### Return type

INTEGER

### Example

The following example returns the number of elements in the array:

```
select json_array_length('[11,12,13,{"f1":21,"f2":[25,26]},14]');
```

```
json_array_length

5
```

The following example returns an error because the JSON is invalid.

```
select json_array_length('[11,12,13,{"f1":21,"f2":[25,26]},14');

An error occurred when executing the SQL command:
select json_array_length('[11,12,13,{"f1":21,"f2":[25,26]},14')
```

The following example sets *null\_if\_invalid* to *true*, so the statement returns NULL instead of returning an error for invalid JSON.

```
select json_array_length('[11,12,13,{"f1":21,"f2":[25,26]},14',true);

json_array_length

```

## JSON\_EXTRACT\_ARRAY\_ELEMENT\_TEXT function

The `JSON_EXTRACT_ARRAY_ELEMENT_TEXT` function returns a JSON array element in the outermost array of a JSON string, using a zero-based index. The first element in an array is at position 0. If the index is negative or out of bound, `JSON_EXTRACT_ARRAY_ELEMENT_TEXT` returns empty string. If the *null\_if\_invalid* argument is set to *true* and the JSON string is invalid, the function returns NULL instead of returning an error.

For more information, see [JSON functions \(p. 1142\)](#).

### Syntax

```
json_extract_array_element_text('json_string', pos [, null_if_invalid])
```

### Arguments

*json\_string*

A properly formatted JSON string.

*pos*

An integer representing the index of the array element to be returned, using a zero-based array index.

*null\_if\_invalid*

A Boolean value that specifies whether to return NULL if the input JSON string is invalid instead of returning an error. To return NULL if the JSON is invalid, specify *true* (t). To return an error if the JSON is invalid, specify *false* (f). The default is *false*.

### Return type

A VARCHAR string representing the JSON array element referenced by *pos*.

## Example

The following example returns array element at position 2, which is the third element of a zero-based array index:

```
select json_extract_array_element_text('[111,112,113]', 2);
json_extract_array_element_text

113
```

The following example returns an error because the JSON is invalid.

```
select json_extract_array_element_text('["a",["b",1,[{"c":2,3,null}]]',1);
An error occurred when executing the SQL command:
select json_extract_array_element_text('["a",["b",1,[{"c":2,3,null}]]',1)
```

The following example sets *null\_if\_invalid* to *true*, so the statement returns NULL instead of returning an error for invalid JSON.

```
select json_extract_array_element_text('["a",["b",1,[{"c":2,3,null}]]',1,true);
json_extract_array_element_text

```

## JSON\_EXTRACT\_PATH\_TEXT function

The JSON\_EXTRACT\_PATH\_TEXT function returns the value for the *key:value* pair referenced by a series of path elements in a JSON string. The JSON path can be nested up to five levels deep. Path elements are case-sensitive. If a path element does not exist in the JSON string, JSON\_EXTRACT\_PATH\_TEXT returns an empty string. If the *null\_if\_invalid* argument is set to *true* and the JSON string is invalid, the function returns NULL instead of returning an error.

For more information, see [JSON functions \(p. 1142\)](#).

### Syntax

```
json_extract_path_text('json_string', 'path_elem' [, 'path_elem'[, ...]] [, null_if_invalid
])
```

### Arguments

*json\_string*

A properly formatted JSON string.

*path\_elem*

A path element in a JSON string. One path element is required. Additional path elements can be specified, up to five levels deep.

*null\_if\_invalid*

A Boolean value that specifies whether to return NULL if the input JSON string is invalid instead of returning an error. To return NULL if the JSON is invalid, specify *true* (t). To return an error if the JSON is invalid, specify *false* (f). The default is *false*.

In a JSON string, Amazon Redshift recognizes \n as a newline character and \t as a tab character. To load a backslash, escape it with a backslash (\\\). For more information, see [Escape characters in JSON \(p. 722\)](#).

## Return type

VARCHAR string representing the JSON value referenced by the path elements.

## Example

The following example returns the value for the path 'f4', 'f6'.

```
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}},'f4', 'f6');

json_extract_path_text

star
```

The following example returns an error because the JSON is invalid.

```
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}},'f4', 'f6');

An error occurred when executing the SQL command:
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}},'f4', 'f6')
```

The following example sets *null\_if\_invalid* to *true*, so the statement returns NULL for invalid JSON instead of returning an error.

```
select json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}},'f4', 'f6',true);

json_extract_path_text

NULL
```

The following example returns the value for the path 'farm', 'barn', 'color', where the value retrieved is at the third level. This sample is formatted with a JSON lint tool, to make it easier to read.

```
select json_extract_path_text('{
 "farm": {
 "barn": {
 "color": "red",
 "feed stocked": true
 }
 }
}', 'farm', 'barn', 'color');

json_extract_path_text

red
```

The following example returns NULL because the 'color' element is missing. This sample is formatted with a JSON lint tool.

```
select json_extract_path_text('{
 "farm": {
 "barn": {}
 }
}', 'farm', 'barn', 'color');
```

```
json_extract_path_text

NULL
```

If the JSON is valid, trying to extract an element that's missing returns NULL.

The following example returns the value for the path 'house', 'appliances', 'washing machine', 'brand'.

```
select json_extract_path_text('{
 "house": {
 "address": {
 "street": "123 Any St.",
 "city": "Any Town",
 "state": "FL",
 "zip": "32830"
 },
 "bathroom": {
 "color": "green",
 "shower": true
 },
 "appliances": {
 "washing machine": {
 "brand": "Any Brand",
 "color": "beige"
 },
 "dryer": {
 "brand": "Any Brand",
 "color": "white"
 }
 }
}', 'house', 'appliances', 'washing machine', 'brand');

json_extract_path_text

Any Brand
```

## JSON\_PARSE function

The JSON\_PARSE function parses data in JSON format and converts it into the SUPER representation.

To ingest into SUPER data type using the INSERT or UPDATE command, use the JSON\_PARSE function. When you use JSON\_PARSE() to parse JSON strings into SUPER values, certain restrictions apply. For additional information, see [Parsing options for SUPER \(p. 387\)](#).

### Syntax

```
JSON_PARSE(json_string)
```

### Arguments

*json\_string*

An expression that returns serialized JSON as a varbyte or varchar type.

### Return type

SUPER

## Example

The following example is an example of the JSON\_PARSE function.

```
SELECT JSON_PARSE('[10001,10002,"abc"]');
 json_parse

[10001,10002,"abc"]
(1 row)
```

```
SELECT JSON_TYPEOF(JSON_PARSE('[10001,10002,"abc"]'));
 json_typeof

array
(1 row)
```

## CAN\_JSON\_PARSE function

The CAN\_JSON\_PARSE function parses data in JSON format and returns true if the result can be converted to a SUPER value using the JSON\_PARSE function.

### Syntax

```
CAN_JSON_PARSE(json_string)
```

### Arguments

*json\_string*

An expression that returns serialized JSON in the varbyte or varchar form.

### Return type

Boolean

## Example

The following example is an example of the CAN\_JSON\_PARSE function.

```
SELECT CAN_JSON_PARSE('[10001,10002,"abc"]');
```

```
can_json_parse

t
```

## JSON\_SERIALIZE function

The JSON\_SERIALIZE function serializes a SUPER expression into textual JSON representation to follow RFC 8259. For more information on that RFC, see [The JavaScript Object Notation \(JSON\) Data Interchange Format](#).

The SUPER size limit is approximately the same as the block limit, and the varchar limit is smaller than the SUPER size limit. Therefore, the JSON\_SERIALIZE function returns an error when the JSON

format exceeds the varchar limit of the system. To check the size of a SUPER expression, see the [JSON\\_SIZE \(p. 1333\)](#) function.

## Syntax

```
JSON_SERIALIZE(super_expression)
```

## Arguments

*super\_expression*

A super expression or column.

## Return type

varchar

## Example

The following example serializes a SUPER value to a string.

```
SELECT JSON_SERIALIZE(JSON_PARSE('[10001,10002,"abc"]'));
 json_serialize

[10001,10002,"abc"]
(1 row)
```

## JSON\_SERIALIZE\_TO\_VARBYTE function

The JSON\_SERIALIZE\_TO\_VARBYTE function converts a SUPER value to a JSON string similar to JSON\_SERIALIZE(), but stored in a VARBYTE value instead.

## Syntax

```
JSON_SERIALIZE_TO_VARBYTE(super_expression)
```

## Arguments

*super\_expression*

A super expression or column.

## Return type

varbyte

## Example

The following example serializes a SUPER value and returns the result in VARBYTE format.

```
SELECT JSON_SERIALIZE_TO_VARBYTE(JSON_PARSE('[10001,10002,"abc"]'));
```

```
json_serialize_to_varbyte
```

```

5b31303030312c31303030322c22616263225d
```

The following example serializes a SUPER value and casts the result to VARCHAR format.

```
SELECT JSON_SERIALIZE_TO_VARBYTE(JSON_PARSE(' [10001,10002,"abc"]'))::VARCHAR;
```

```

json_serialize_to_varbyte

[10001,10002,"abc"]
```

## Machine learning functions

By using Amazon Redshift machine learning (ML), you can train ML models using SQL statements and invoke them in SQL queries for prediction. Amazon Redshift model explainability includes feature-importance values to help you understand how each attribute in your training data contributes to the predicted result.

Following, you can find descriptions for the machine learning functions for SQL that Amazon Redshift supports.

### Topics

- [EXPLAIN\\_MODEL function \(p. 1152\)](#)

## EXPLAIN\_MODEL function

The EXPLAIN\_MODEL function returns a SUPER data type that contains a model explainability report in a JSON format. The explainability report contains information about the Shapley value for all model features.

The EXPLAIN\_MODEL function currently supports only the AUTO ON or AUTO OFF XGBoost models.

When the explainability report isn't available, the function returns statuses showing on the progress of the model. These include Waiting for training job to complete, Waiting for processing job to complete, and Processing job failed.

When you run the CREATE MODEL statement, the explanation state becomes Waiting for training job to complete. When the model has been trained and an explanation request is sent, the explanation state becomes Waiting for processing job to complete. When the model explanation completes successfully, the full explainability report is available. Otherwise, the state becomes Processing job failed.

### Syntax

```
EXPLAIN_MODEL ('schema_name.model_name')
```

### Argument

*schema\_name*

The name of the schema. If no schema\_name is specified, then the current schema is selected.

*model\_name*

The name of the model. The model name in a schema must be unique.

## Return type

The EXPLAIN\_MODEL function returns a SUPER data type, as shown following.

```
{"version": "1.0", "explanations": {"kernel_shap": {"label0": {"global_shap_values": [{"x0": 0.05, "x1": 0.1, "x2": 0.3, "x3": 0.15}, "expected_value": 0.5]} }}}}
```

## Examples

The following example returns the explanation state waiting for training job to complete.

```
select explain_model('customer_churn_auto_model');
 explain_model

>{"explanations":"waiting for training job to complete"}
(1 row)
```

When the model explanation completes successfully, the full explainability report is available as follows.

Because the EXPLAIN\_MODEL function returns the SUPER data type, you can query the explainability report. By doing this, you can extract global\_shap\_values, expected\_value, or feature-specific Shapley values.

The following example extracts `global_shap_values` for the model.

```
select json_table.report.explanations.kernel_shap.label0.global_shap_values from (select
 explain_model('customer_churn_auto_model') as report) as json_table;
 global_shap_values

>{"state":0.10983770427197151,"account_length":0.1772441398408543,"area_code":0.08626823968639591,"phone"
(1 row)
```

The following example extracts global `shape` values for the feature `x0`.

```
select json_table.report.explanations.kernel_shap.label0.global_shap_values.x0 from (select
 explain_model('customer_churn_auto_model') as report) as json_table;
x0

 0.05386043365892927
(1 row)
```

If the model is created in a specific schema and you have access to the created model, then you can query the model explanation as shown following.

```
-- Check the current schema
SHOW search_path;
 search_path

$user, public
```

```
(1 row)
-- If you have the privilege to access the model explanation
-- in `test_schema`
SELECT explain_model('test_schema.test_model_name');
 explain_model

>{"explanations":"waiting for training job to complete"}
(1 row)
```

## Math functions

### Topics

- [Mathematical operator symbols \(p. 1154\)](#)
- [ABS function \(p. 1156\)](#)
- [ACOS function \(p. 1157\)](#)
- [ASIN function \(p. 1157\)](#)
- [ATAN function \(p. 1158\)](#)
- [ATAN2 function \(p. 1159\)](#)
- [CBRT function \(p. 1159\)](#)
- [CEILING \(or CEIL\) function \(p. 1160\)](#)
- [COS function \(p. 1161\)](#)
- [COT function \(p. 1161\)](#)
- [DEGREES function \(p. 1162\)](#)
- [DEXP function \(p. 1162\)](#)
- [DLOG1 function \(p. 1163\)](#)
- [DLOG10 function \(p. 1163\)](#)
- [EXP function \(p. 1164\)](#)
- [FLOOR function \(p. 1164\)](#)
- [LN function \(p. 1165\)](#)
- [LOG function \(p. 1166\)](#)
- [MOD function \(p. 1167\)](#)
- [PI function \(p. 1168\)](#)
- [POWER function \(p. 1169\)](#)
- [RADIANS function \(p. 1170\)](#)
- [RANDOM function \(p. 1170\)](#)
- [ROUND function \(p. 1172\)](#)
- [SIN function \(p. 1173\)](#)
- [SIGN function \(p. 1174\)](#)
- [SQRT function \(p. 1175\)](#)
- [TAN function \(p. 1176\)](#)
- [TRUNC function \(p. 1177\)](#)

This section describes the mathematical operators and functions supported in Amazon Redshift.

### Mathematical operator symbols

The following table lists the supported mathematical operators.

## Supported operators

| Operator | Description         | Example   | Result |
|----------|---------------------|-----------|--------|
| +        | addition            | 2 + 3     | 5      |
| -        | subtraction         | 2 - 3     | -1     |
| *        | multiplication      | 2 * 3     | 6      |
| /        | division            | 4 / 2     | 2      |
| %        | modulo              | 5 % 4     | 1      |
| ^        | exponentiation      | 2.0 ^ 3.0 | 8      |
|          | square root         | / 25.0    | 5      |
| /        | cube root           | / / 27.0  | 3      |
| @        | absolute value      | @ -5.0    | 5      |
| <<       | bitwise shift left  | 1 << 4    | 16     |
| >>       | bitwise shift right | 8 >> 2    | 2      |
| &        | bitwise and         | 8 & 2     | 0      |

## Examples

Calculate the commission paid plus a \$2.00 handling for a given transaction:

```
select commission, (commission + 2.00) as comm
from sales where salesid=10000;

commission | comm
-----+-----
28.05 | 30.05
(1 row)
```

Calculate 20 percent of the sales price for a given transaction:

```
select pricepaid, (pricepaid * .20) as twentypct
from sales where salesid=10000;

pricepaid | twentypct
-----+-----
187.00 | 37.400
(1 row)
```

Forecast ticket sales based on a continuous growth pattern. In this example, the subquery returns the number of tickets sold in 2008. That result is multiplied exponentially by a continuous growth rate of 5% over 10 years.

```
select (select sum(qtysold) from sales, date
 where sales.dateid=date.dateid and year=2008)
 ^ ((5::float/100)*10) as qty10years;
```

```
qty10years

587.664019657491
(1 row)
```

Find the total price paid and commission for sales with a date ID that is greater than or equal to 2000. Then subtract the total commission from the total price paid.

```
select sum (pricepaid) as sum_price, dateid,
sum (commission) as sum_comm, (sum (pricepaid) - sum (commission)) as value
from sales where dateid >= 2000
group by dateid order by dateid limit 10;

sum_price | dateid | sum_comm | value
-----+-----+-----+-----
364445.00 | 2044 | 54666.75 | 309778.25
349344.00 | 2112 | 52401.60 | 296942.40
343756.00 | 2124 | 51563.40 | 292192.60
378595.00 | 2116 | 56789.25 | 321805.75
328725.00 | 2080 | 49308.75 | 279416.25
349554.00 | 2028 | 52433.10 | 297120.90
249207.00 | 2164 | 37381.05 | 211825.95
285202.00 | 2064 | 42780.30 | 242421.70
320945.00 | 2012 | 48141.75 | 272803.25
321096.00 | 2016 | 48164.40 | 272931.60
(10 rows)
```

## ABS function

ABS calculates the absolute value of a number, where that number can be a literal or an expression that evaluates to a number.

### Syntax

```
ABS (number)
```

### Arguments

*number*

Number or expression that evaluates to a number. It can be the SMALLINT, INTEGER, BIGINT, DECIMAL, FLOAT4, FLOAT8, or SUPER type.

### Return type

ABS returns the same data type as its argument.

### Examples

Calculate the absolute value of -38:

```
select abs (-38);
abs

38
(1 row)
```

Calculate the absolute value of (14-76):

```
select abs (14-76);
abs

62
(1 row)
```

## ACOS function

ACOS is a trigonometric function that returns the arc cosine of a number. The return value is in radians and is between 0 and PI.

### Syntax

```
ACOS(number)
```

### Arguments

*number*

The input parameter is a double precision number.

### Return type

The ACOS function returns a double precision number.

### Examples

The following example returns the arc cosine of -1:

```
select acos(-1);
acos

3.14159265358979
(1 row)
```

The following example converts the arc cosine of .5 to the equivalent number of degrees:

```
select (acos(.5) * 180/(select pi())) as degrees;
degrees

60
(1 row)
```

## ASIN function

ASIN is a trigonometric function that returns the arc sine of a number. The return value is in radians and is between PI/2 and -PI/2.

### Syntax

```
ASIN(number)
```

## Argument

*number*

The input parameter is a double precision number.

## Return type

The ASIN function returns a double precision number.

## Examples

The following example returns the arc sine of 1 and multiples it by 2:

```
select asin(1)*2 as pi;
pi

3.14159265358979
(1 row)
```

The following example converts the arc sine of .5 to the equivalent number of degrees:

```
select (asin(.5) * 180/(select pi())) as degrees;
degrees

30
(1 row)
```

## ATAN function

ATAN is a trigonometric function that returns the arc tangent of a number. The return value is in radians and is between -PI and PI.

## Syntax

```
ATAN(number)
```

## Argument

*number*

The input parameter is a double precision number.

## Return type

The ATAN function returns a double precision number.

## Examples

The following example returns the arc tangent of 1 and multiplies it by 4:

```
select atan(1) * 4 as pi;
pi

3.14159265358979
(1 row)
```

The following example converts the arc tangent of 1 to the equivalent number of degrees:

```
select (atan(1) * 180/(select pi())) as degrees;
degrees

45
(1 row)
```

## ATAN2 function

ATAN2 is a trigonometric function that returns the arc tangent of a one number divided by another number. The return value is in radians and is between PI/2 and -PI/2.

### Syntax

```
ATAN2(number1, number2)
```

### Arguments

*number1*

The first input parameter is a double precision number.

*number2*

The second parameter is a double precision number.

### Return type

The ATAN2 function returns a double precision number.

### Examples

The following example returns the arc tangent of 2/2 and multiplies it by 4:

```
select atan2(2,2) * 4 as pi;
pi

3.14159265358979
(1 row)
```

The following example converts the arc tangent of 1/0 (or 0) to the equivalent number of degrees:

```
select (atan2(1,0) * 180/(select pi())) as degrees;
degrees

90
(1 row)
```

## CBRT function

The CBRT function is a mathematical function that calculates the cube root of a number.

### Syntax

```
CBRT (number)
```

## Argument

CBRT takes a DOUBLE PRECISION number as an argument.

## Return type

CBRT returns a DOUBLE PRECISION number.

## Examples

Calculate the cube root of the commission paid for a given transaction:

```
select cbrt(commission) from sales where salesid=10000;

cbrt

3.03839539048843
(1 row)
```

## CEILING (or CEIL) function

The CEILING or CEIL function is used to round a number up to the next whole number. (The [FLOOR function \(p. 1164\)](#) rounds a number down to the next whole number.)

## Syntax

```
CEIL | CEILING(number)
```

## Arguments

### *number*

The number or expression that evaluates to a number. It can be the SMALLINT, INTEGER, BIGINT, DECIMAL, FLOAT4, FLOAT8, or SUPER type.

## Return type

CEILING and CEIL return the same data type as its argument.

When the input is of the SUPER type, the output retains the same dynamic type as the input while the static type remains the SUPER type. When the dynamic type of SUPER isn't a number, Amazon Redshift returns a null.

## Example

Calculate the ceiling of the commission paid for a given sales transaction:

```
select ceiling(commission) from sales
where salesid=10000;

ceiling

29
(1 row)
```

## COS function

COS is a trigonometric function that returns the cosine of a number. The return value is in radians and is between -1 and 1, inclusive.

### Syntax

```
COS(double_precision)
```

### Argument

*number*

The input parameter is a double precision number.

### Return type

The COS function returns a double precision number.

### Examples

The following example returns cosine of 0:

```
select cos(0);
cos

1
(1 row)
```

The following example returns the cosine of PI:

```
select cos(pi());
cos

-1
(1 row)
```

## COT function

COT is a trigonometric function that returns the cotangent of a number. The input parameter must be nonzero.

### Syntax

```
COT(number)
```

### Argument

*number*

The input parameter is a double precision number.

### Return type

The COT function returns a double precision number.

## Examples

The following example returns the cotangent of 1:

```
select cot(1);
cot

0.642092615934331
(1 row)
```

## DEGREES function

Converts an angle in radians to its equivalent in degrees.

### Syntax

```
DEGREES(number)
```

### Argument

*number*

The input parameter is a double precision number.

### Return type

The DEGREES function returns a double precision number.

### Examples

The following example returns the degree equivalent of .5 radians:

```
select degrees(.5);
degrees

28.6478897565412
(1 row)
```

The following example converts PI radians to degrees:

```
select degrees(pi());
degrees

180
(1 row)
```

## DEXP function

The DEXP function returns the exponential value in scientific notation for a double precision number. The only difference between the DEXP and EXP functions is that the parameter for DEXP must be a double precision.

### Syntax

```
DEXP(number)
```

## Argument

*number*

The input parameter is a double precision number.

## Return type

The DEXP function returns a double precision number.

## Example

Use the DEXP function to forecast ticket sales based on a continuous growth pattern. In this example, the subquery returns the number of tickets sold in 2008. That result is multiplied by the result of the DEXP function, which specifies a continuous growth rate of 7% over 10 years.

```
select (select sum(qtysold) from sales, date
 where sales.dateid=date.dateid
 and year=2008) * dexp((7::float/100)*10) qty2010;
qty2010

695447.483772222
(1 row)
```

## DLOG1 function

The DLOG1 function returns the natural logarithm of the input parameter. Synonym for the LN function.  
of [LN function \(p. 1165\)](#).

## DLOG10 function

The DLOG10 returns the base 10 logarithm of the input parameter. Synonym of the LOG function.

Synonym of [LOG function \(p. 1166\)](#).

## Syntax

```
DLOG10(number)
```

## Argument

*number*

The input parameter is a double precision number.

## Return type

The DLOG10 function returns a double precision number.

## Example

The following example returns the base 10 logarithm of the number 100:

```
select dlog10(100);
```

```
dlog10

2
(1 row)
```

## EXP function

The EXP function implements the exponential function for a numeric expression, or the base of the natural logarithm, e, raised to the power of expression. The EXP function is the inverse of [LN function \(p. 1165\)](#).

### Syntax

```
EXP (expression)
```

### Argument

*expression*

The expression must be an INTEGER, DECIMAL, or DOUBLE PRECISION data type.

### Return type

EXP returns a DOUBLE PRECISION number.

### Example

Use the EXP function to forecast ticket sales based on a continuous growth pattern. In this example, the subquery returns the number of tickets sold in 2008. That result is multiplied by the result of the EXP function, which specifies a continuous growth rate of 7% over 10 years.

```
select (select sum(qtysold) from sales, date
 where sales.dateid=date.dateid
 and year=2008) * exp((7::float/100)*10) qty2018;
qty2018

695447.483772222
(1 row)
```

## FLOOR function

The FLOOR function rounds a number down to the next whole number.

### Syntax

```
FLOOR (number)
```

### Argument

*number*

The number or expression that evaluates to a number. It can be the SMALLINT, INTEGER, BIGINT, DECIMAL, FLOAT4, FLOAT8, or SUPER type.

## Return type

FLOOR returns the same data type as its argument.

When the input is of the SUPER type, the output retains the same dynamic type as the input while the static type remains the SUPER type. When the dynamic type of SUPER isn't a number, Amazon Redshift returns a null.

## Example

The example shows the value of the commission paid for a given sales transaction before and after using the FLOOR function.

```
select commission from sales
where salesid=10000;

floor

28.05
(1 row)

select floor(commission) from sales
where salesid=10000;

floor

28
(1 row)
```

## LN function

Returns the natural logarithm of the input parameter. Synonym of the DLOG1 function.

Synonym of [DLOG1 function \(p. 1163\)](#).

## Syntax

```
LN(expression)
```

## Argument

*expression*

The target column or expression that the function operates on.

### Note

This function returns an error for some data types if the expression references an Amazon Redshift user-created table or an Amazon Redshift STL or STV system table.

Expressions with the following data types produce an error if they reference a user-created or system table. Expressions with these data types run exclusively on the leader node:

- BOOLEAN
- CHAR
- DATE
- DECIMAL or NUMERIC
- TIMESTAMP
- VARCHAR

Expressions with the following data types run successfully on user-created tables and STL or STV system tables:

- BIGINT
- DOUBLE PRECISION
- INTEGER
- REAL
- SMALLINT

## Return type

The LN function returns the same type as the expression.

## Example

The following example returns the natural logarithm, or base e logarithm, of the number 2.718281828:

```
select ln(2.718281828);
ln

0.9999999998311267
(1 row)
```

Note that the answer is nearly equal to 1.

This example returns the natural logarithm of the values in the USERID column in the USERS table:

```
select username, ln(userid) from users order by userid limit 10;

username | ln
-----+-----
JSG99FHE | 0
PGL08LJI | 0.693147180559945
IFT66TXU | 1.09861228866811
XDZ38RDD | 1.38629436111989
AEB55QTM | 1.6094379124341
NDQ15VBM | 1.79175946922805
OWY35QYB | 1.94591014905531
AZG78YIP | 2.07944154167984
MSD36KVR | 2.19722457733622
WKW41AIW | 2.30258509299405
(10 rows)
```

## LOG function

Returns the base 10 logarithm of a number.

Synonym of [DLOG10 function \(p. 1163\)](#).

## Syntax

```
LOG(number)
```

## Argument

*number*

The input parameter is a double precision number.

## Return type

The LOG function returns a double precision number.

## Example

The following example returns the base 10 logarithm of the number 100:

```
select log(100);
dlog10

2
(1 row)
```

## MOD function

Returns the remainder of two numbers, otherwise known as a *modulo* operation. To calculate the result, the first parameter is divided by the second.

## Syntax

```
MOD(number1, number2)
```

## Arguments

*number1*

The first input parameter is an INTEGER, SMALLINT, BIGINT, or DECIMAL number. If either parameter is a DECIMAL type, the other parameter must also be a DECIMAL type. If either parameter is an INTEGER, the other parameter can be an INTEGER, SMALLINT, or BIGINT. Both parameters can also be SMALLINT or BIGINT, but one parameter cannot be a SMALLINT if the other is a BIGINT.

*number2*

The second parameter is an INTEGER, SMALLINT, BIGINT, or DECIMAL number. The same data type rules apply to *number2* as to *number1*.

## Return type

Valid return types are DECIMAL, INT, SMALLINT, and BIGINT. The return type of the MOD function is the same numeric type as the input parameters, if both input parameters are the same type. If either input parameter is an INTEGER, however, the return type will also be an INTEGER.

## Usage notes

You can use % as a modulo operator.

## Examples

Some examples use the *TICKIT* sample data set. For more information, see [Sample database](#).

The following example return the remainder when a number is divided by another:

```
SELECT MOD(10, 4);
```

```
mod

2
```

The following example returns a decimal result:

```
SELECT MOD(10.5, 4);

mod

2.5
```

You can cast parameter values:

```
SELECT MOD(CAST(16.4 as integer), 5);

mod

1
```

Check if the first parameter is even by dividing it by 2:

```
SELECT mod(5,2) = 0 as is_even;

is_even

false
```

You can use the % as a modulo operator:

```
SELECT 11 % 4 as remainder;

remainder

3
```

The following example returns information for odd-numbered categories in the CATEGORY table:

```
select catid, catname
from category
where mod(catid,2)=1
order by 1,2;

catid | catname
-----+-----
1 | MLB
3 | NFL
5 | MLS
7 | Plays
9 | Pop
11 | Classical
(6 rows)
```

## PI function

The PI function returns the value of PI to 14 decimal places.

## Syntax

```
PI()
```

## Return type

PI returns a DOUBLE PRECISION number.

## Examples

Return the value of pi:

```
select pi();

pi

3.14159265358979
(1 row)
```

## POWER function

### Syntax

The POWER function is an exponential function that raises a numeric expression to the power of a second numeric expression. For example, 2 to the third power is calculated as power(2,3), with a result of 8.

```
POW | POWER (expression1, expression2)
```

POW and POWER are synonyms.

### Arguments

#### *expression1*

Numeric expression to be raised. Must be an integer, decimal, or floating-point data type.

#### *expression2*

Power to raise *expression1*. Must be an integer, decimal, or floating-point data type.

## Return type

POWER returns a DOUBLE PRECISION number.

## Examples

In the following example, the POWER function is used to forecast what ticket sales will look like in the next 10 years, based on the number of tickets sold in 2008 (the result of the subquery). The growth rate is set at 7% per year in this example.

```
select (select sum(qtysold) from sales, date
where sales.dateid=date.dateid
and year=2008) * pow((1+7::float/100),10) qty2010;

qty2010
```

```

679353.754088594
(1 row)
```

The following example is a variation on the previous example, with the growth rate at 7% per year but the interval set to months (120 months over 10 years):

```
select (select sum(qtysold) from sales, date
where sales.dateid=date.dateid
and year=2008) * pow((1+7::float/100/12),120) qty2010;

qty2010

694034.54678046
(1 row)
```

## RADIANS function

Converts an angle in degrees to its equivalent in radians.

### Syntax

```
RADIANS(number)
```

### Argument

*string*

The input parameter is a double precision number.

### Return type

The RADIANS function returns a double precision number.

### Examples

The following example returns the radian equivalent of 180 degrees:

```
select radians(180);
radians

3.14159265358979
(1 row)
```

## RANDOM function

The RANDOM function generates a random value between 0.0 (inclusive) and 1.0 (exclusive).

### Syntax

```
RANDOM()
```

### Return type

RANDOM returns a DOUBLE PRECISION number.

## Usage notes

Call RANDOM after setting a seed value with the [SET \(p. 977\)](#) command to cause RANDOM to generate numbers in a predictable sequence.

## Examples

1. Compute a random value between 0 and 99. If the random number is 0 to 1, this query produces a random number from 0 to 100:

```
select cast (random() * 100 as int);
int4

24
(1 row)
```

2. Retrieve a uniform random sample of 10 items:

```
select *
from sales
order by random()
limit 10;
```

Now retrieve a random sample of 10 items, but choose the items in proportion to their prices. For example, an item that is twice the price of another would be twice as likely to appear in the query results:

```
select *
from sales
order by log(1 - random()) / pricepaid
limit 10;
```

3. This example uses the [SET \(p. 977\)](#) command to set a SEED value so that RANDOM generates a predictable sequence of numbers.

First, return three RANDOM integers without setting the SEED value first:

```
select cast (random() * 100 as int);
int4

6
(1 row)

select cast (random() * 100 as int);
int4

68
(1 row)

select cast (random() * 100 as int);
int4

56
(1 row)
```

Now, set the SEED value to .25, and return three more RANDOM numbers:

```
set seed to .25;
```

```

select cast (random() * 100 as int);
int4

21
(1 row)

select cast (random() * 100 as int);
int4

79
(1 row)

select cast (random() * 100 as int);
int4

12
(1 row)

```

Finally, reset the SEED value to .25, and verify that RANDOM returns the same results as the previous three calls:

```

set seed to .25;
select cast (random() * 100 as int);
int4

21
(1 row)

select cast (random() * 100 as int);
int4

79
(1 row)

select cast (random() * 100 as int);
int4

12
(1 row)

```

## ROUND function

The ROUND function rounds numbers to the nearest integer or decimal.

The ROUND function can optionally include a second argument as an integer to indicate the number of decimal places for rounding, in either direction. When you don't provide the second argument, the function rounds to the nearest whole number. When the second argument  $>n$  is specified, the function rounds to the nearest number with  $n$  decimal places of precision.

### Syntax

```
ROUND (number [, integer])
```

### Argument

*number*

A number or expression that evaluates to a number. It can be the DECIMAL, FLOAT8 or SUPER type. Amazon Redshift can convert other data types per the implicit conversion rules.

*integer* (optional)

An integer that indicates the number of decimal places for rounding in either directions. The extra argument isn't supported for the SUPER data type.

## Return type

ROUND returns the same numeric data type as the input argument(s).

When the input is of the SUPER type, the output retains the same dynamic type as the input while the static type remains the SUPER type. When the dynamic type of SUPER isn't a number, Amazon Redshift returns a null.

## Examples

Round the commission paid for a given transaction to the nearest whole number.

```
select commission, round(commission)
from sales where salesid=10000;

commission | round
-----+-----
 28.05 | 28
(1 row)
```

Round the commission paid for a given transaction to the first decimal place.

```
select commission, round(commission, 1)
from sales where salesid=10000;

commission | round
-----+-----
 28.05 | 28.1
(1 row)
```

For the same query, extend the precision in the opposite direction.

```
select commission, round(commission, -1)
from sales where salesid=10000;

commission | round
-----+-----
 28.05 | 30
(1 row)
```

## SIN function

SIN is a trigonometric function that returns the sine of a number. The return value is between -1 and 1.

### Syntax

```
SIN(number)
```

### Argument

*number*

A double precision number in radians.

## Return type

The SIN function returns a double precision number.

## Examples

The following example returns the sine of -PI:

```
select sin(-pi());

sin

-1.22464679914735e-16
(1 row)
```

## SIGN function

The SIGN function returns the sign (positive or negative) of a number. The result of the SIGN function is 1, -1, or 0 indicating the sign of the argument.

## Syntax

```
SIGN (number)
```

## Argument

*number*

Number or expression that evaluates to a number. It can be the DECIMAL, FLOAT8, or SUPER type. Amazon Redshift can convert other data types per the implicit conversion rules.

## Return type

SIGN returns the same numeric data type as the input argument(s). If the input is DECIMAL, the output is DECIMAL(1,0).

When the input is of the SUPER type, the output retains the same dynamic type as the input while the static type remains the SUPER type. When the dynamic type of SUPER isn't a number, Amazon Redshift returns a null.

## Examples

Determine the sign of the commission paid for a given transaction:

```
select commission, sign (commission)
from sales where salesid=10000;

commission | sign
-----+-----
 28.05 | 1
(1 row)
```

The following example shows that "t2.d" has double precision as its type since the input is double precision and that "t2.n" has numeric(1,0) as output since the input is numeric.

```
CREATE TABLE t1(d double precision, n numeric(12, 2));
INSERT INTO t1 VALUES (4.25, 4.25), (-4.25, -4.25);
CREATE TABLE t2 AS SELECT SIGN(d) AS d, SIGN(n) AS n FROM t1;
\d t1
\d t2
```

The output of the “\d” commands is:

| Table "public.t1" |                  |          |         |         |         |            |           |
|-------------------|------------------|----------|---------|---------|---------|------------|-----------|
| Column            | Type             | Encoding | DistKey | SortKey | Preload | Encryption | Modifiers |
| d                 | double precision | none     | f       | 0       | f       | none       |           |
| n                 | numeric(12,2)    | lzo      | f       | 0       | f       | none       |           |

| Table "public.t2" |                  |          |         |         |         |            |           |
|-------------------|------------------|----------|---------|---------|---------|------------|-----------|
| Column            | Type             | Encoding | DistKey | SortKey | Preload | Encryption | Modifiers |
| d                 | double precision | none     | f       | 0       | f       | none       |           |
| n                 | numeric(1,0)     | lzo      | f       | 0       | f       | none       |           |

## SQRT function

The SQRT function returns the square root of a numeric value. The square root is a number multiplied by itself to get the given value.

### Syntax

```
SQRT (expression)
```

### Argument

*expression*

The expression must have an integer, decimal, or floating-point data type. The expression can include functions. The system might perform implicit type conversions.

### Return type

SQRT returns a DOUBLE PRECISION number.

### Examples

The following example returns the square root of a number.

```
select sqrt(16);

sqrt

4
```

The following example performs an implicit type conversion.

```
select sqrt('16');

sqrt
```

```

4
```

The following example nests functions to perform a more complex task.

```
select sqrt(round(16.4));

sqrt

4
```

The following example results in the length of the radius when given the area of a circle. It calculates the radius in inches, for instance, when given the area in square inches. The area in the sample is 20.

```
select sqrt(20/pi());
```

This returns the value 5.046265044040321.

The following example returns the square root for COMMISSION values from the SALES table. The COMMISSION column is a DECIMAL column. This example shows how you can use the function in a query with more complex conditional logic.

```
select sqrt(commission)
from sales where salesid < 10 order by salesid;

sqrt

10.4498803820905
3.37638860322683
7.24568837309472
5.1234753829798
...
```

The following query returns the rounded square root for the same set of COMMISSION values.

```
select salesid, commission, round(sqrt(commission))
from sales where salesid < 10 order by salesid;

salesid | commission | round
-----+-----+-----
1 | 109.20 | 10
2 | 11.40 | 3
3 | 52.50 | 7
4 | 26.25 | 5
...
```

For more information about sample data in Amazon Redshift, see [Sample database](#).

## TAN function

TAN is a trigonometric function that returns the tangent of a number. The input parameter must be a non-zero number (in radians).

### Syntax

```
TAN(number)
```

## Argument

*number*

The input parameter is a double precision number.

## Return type

The TAN function returns a double precision number.

## Examples

The following example returns the tangent of 0:

```
select tan(0);
tan

0
(1 row)
```

## TRUNC function

The TRUNC function truncates numbers to the previous integer or decimal.

The TRUNC function can optionally include a second argument as an integer to indicate the number of decimal places for rounding, in either direction. When you don't provide the second argument, the function rounds to the nearest whole number. When the second argument >*n* is specified, the function rounds to the nearest number with >*n* decimal places of precision. This function also truncates a timestamp and returns a date.

## Syntax

```
TRUNC (number [, integer] |
timestamp)
```

## Arguments

*number*

A number or expression that evaluates to a number. It can be the DECIMAL, FLOAT8 or SUPER type. Amazon Redshift can convert other data types per the implicit conversion rules.

*integer* (optional)

An integer that indicates the number of decimal places of precision, in either direction. If no integer is provided, the number is truncated as a whole number; if an integer is specified, the number is truncated to the specified decimal place. This isn't supported for the SUPER data type.

*timestamp*

The function can also return the date from a timestamp. (To return a timestamp value with 00:00:00 as the time, cast the function result to a timestamp.)

## Return type

TRUNC returns the same data type as the first input argument. For timestamps, TRUNC returns a date.

When the input is of the SUPER type, the output retains the same dynamic type as the input while the static type remains the SUPER type. When the dynamic type of SUPER isn't a number, Amazon Redshift returns a null.

## Examples

Truncate the commission paid for a given sales transaction.

```
select commission, trunc(commission)
from sales where salesid=784;

commission | trunc
-----+-----
 111.15 | 111

(1 row)
```

Truncate the same commission value to the first decimal place.

```
select commission, trunc(commission,1)
from sales where salesid=784;

commission | trunc
-----+-----
 111.15 | 111.1

(1 row)
```

Truncate the commission with a negative value for the second argument; 111.15 is rounded down to 110.

```
select commission, trunc(commission,-1)
from sales where salesid=784;

commission | trunc
-----+-----
 111.15 | 110

(1 row)
```

Return the date portion from the result of the SYSDATE function (which returns a timestamp):

```
select sysdate;

timestamp

2011-07-21 10:32:38.248109
(1 row)

select trunc(sysdate);

trunc

2011-07-21
(1 row)
```

Apply the TRUNC function to a TIMESTAMP column. The return type is a date.

```
select trunc(starttime) from event
order by eventid limit 1;
```

```
trunc

2008-01-25
(1 row)
```

## Object functions

Following are the SQL object functions that Amazon Redshift supports to create SUPER type objects:

### Topics

- [object function \(p. 1179\)](#)

## object function

Creates an object of the SUPER data type.

### Syntax

```
OBJECT ([key1, value1] [, key2, value2 ...])
```

### Arguments

*key1, key2*

Expressions that evaluate to VARCHAR type strings.

*value1, value2*

Expressions of any Amazon Redshift data type except datetime types, since Amazon Redshift doesn't cast datetime types to the SUPER data type. For more information on datetime types, see [Datetime types \(p. 585\)](#).

value expressions in an object don't need to be of the same data type.

### Returns

The object function returns the SUPER data type.

### Example

```
--creates an empty object
select object();
object

{}

(1 row)

--creates objects with different keys and values
select object('a', 1, 'b', true, 'c', 3.14);
object

{"a":1,"b":true,"c":3.14}
(1 row)

select object('a', object('aa', 1), 'b', array(2,3), 'c', json_parse('{}'));
```

```
object

{"a":{"aa":1}, "b":[2,3], "c":{}}
(1 row)

--creates objects using columns from a table
create table bar (k varchar, v super);
insert into bar values ('k1', json_parse('[1]')), ('k2', json_parse('{}'));
select object(k, v) from bar;

>{"k1": [1]}
>{"k2": {}}
(2 rows)

--errors out because DATE type can't be converted to SUPER type
select object('k', '2008-12-31::date');
ERROR: OBJECT could not convert type date to super
```

## Spatial functions

Relationships between geometry objects are based on the Dimensionally Extended nine-Intersection Model (DE-9IM). This model defines predicates such as equals, contains, and covers. For more information about the definition of spatial relationships, see [DE-9IM](#) in Wikipedia.

Amazon Redshift provides spatial functions that work with GEOMETRY and GEOGRAPHY data types. The following lists the functions that support the GEOGRAPHY data type:

- [ST\\_Area \(p. 1188\)](#)
- [ST\\_AsEWKT \(p. 1190\)](#)
- [ST\\_AsGeoJSON \(p. 1191\)](#)
- [ST\\_AsHexEWKB \(p. 1194\)](#)
- [ST\\_AsHexWKB \(p. 1193\)](#)
- [ST\\_AsText \(p. 1194\)](#)
- [ST\\_Distance \(p. 1210\)](#)
- [ST\\_GeogFromText \(p. 1221\)](#)
- [ST\\_GeogFromWKB \(p. 1222\)](#)
- [ST\\_Length \(p. 1238\)](#)
- [ST\\_NPoints \(p. 1252\)](#)
- [ST\\_Perimeter \(p. 1256\)](#)

The following lists the full set of spatial functions supported by Amazon Redshift.

### Topics

- [AddBBox \(p. 1183\)](#)
- [DropBBox \(p. 1183\)](#)
- [GeometryType \(p. 1184\)](#)
- [ST\\_AddPoint \(p. 1185\)](#)
- [ST\\_Angle \(p. 1186\)](#)
- [ST\\_Area \(p. 1188\)](#)
- [ST\\_AsBinary \(p. 1189\)](#)
- [ST\\_AsEWKB \(p. 1189\)](#)
- [ST\\_AsEWKT \(p. 1190\)](#)

- [ST\\_AsGeoJSON \(p. 1191\)](#)
- [ST\\_AsHexWKB \(p. 1193\)](#)
- [ST\\_AsHexEWKB \(p. 1194\)](#)
- [ST\\_AsText \(p. 1194\)](#)
- [ST\\_Azimuth \(p. 1196\)](#)
- [ST\\_Boundary \(p. 1196\)](#)
- [ST\\_Buffer \(p. 1197\)](#)
- [ST\\_Centroid \(p. 1199\)](#)
- [ST\\_Collect \(p. 1200\)](#)
- [ST\\_Contains \(p. 1203\)](#)
- [ST\\_ContainsProperly \(p. 1204\)](#)
- [ST\\_ConvexHull \(p. 1204\)](#)
- [ST\\_CoveredBy \(p. 1206\)](#)
- [ST\\_Covers \(p. 1207\)](#)
- [ST\\_Crosses \(p. 1208\)](#)
- [ST\\_Dimension \(p. 1209\)](#)
- [ST\\_Disjoint \(p. 1210\)](#)
- [ST\\_Distance \(p. 1210\)](#)
- [ST\\_DistanceSphere \(p. 1211\)](#)
- [ST\\_DWithin \(p. 1213\)](#)
- [ST\\_EndPoint \(p. 1213\)](#)
- [ST\\_Envelope \(p. 1214\)](#)
- [ST\\_Equals \(p. 1215\)](#)
- [ST\\_ExteriorRing \(p. 1216\)](#)
- [ST\\_Force2D \(p. 1217\)](#)
- [ST\\_Force3D \(p. 1217\)](#)
- [ST\\_Force3DM \(p. 1218\)](#)
- [ST\\_Force3DZ \(p. 1218\)](#)
- [ST\\_Force4D \(p. 1219\)](#)
- [ST\\_GeoHash \(p. 1220\)](#)
- [ST\\_GeogFromText \(p. 1221\)](#)
- [ST\\_GeogFromWKB \(p. 1222\)](#)
- [ST\\_GeometryN \(p. 1222\)](#)
- [ST\\_GeometryType \(p. 1223\)](#)
- [ST\\_GeomFromEWKB \(p. 1224\)](#)
- [ST\\_GeomFromEWKT \(p. 1225\)](#)
- [ST\\_GeomFromGeoJSON \(p. 1226\)](#)
- [ST\\_GeomFromGeoHash \(p. 1227\)](#)
- [ST\\_GeomFromText \(p. 1228\)](#)
- [ST\\_GeomFromWKB \(p. 1229\)](#)
- [ST\\_InteriorRingN \(p. 1230\)](#)
- [ST\\_Intersects \(p. 1231\)](#)
- [ST\\_Intersection \(p. 1232\)](#)
- [ST\\_IsPolygonCCW \(p. 1233\)](#)
- [ST\\_IsPolygonCW \(p. 1234\)](#)

- [ST\\_IsClosed \(p. 1234\)](#)
- [ST\\_IsCollection \(p. 1235\)](#)
- [ST\\_IsEmpty \(p. 1236\)](#)
- [ST\\_IsRing \(p. 1236\)](#)
- [ST\\_IsSimple \(p. 1237\)](#)
- [ST\\_IsValid \(p. 1237\)](#)
- [ST\\_Length \(p. 1238\)](#)
- [ST\\_LengthSphere \(p. 1239\)](#)
- [ST\\_Length2D \(p. 1240\)](#)
- [ST\\_LineFromMultiPoint \(p. 1240\)](#)
- [ST\\_LineInterpolatePoint \(p. 1241\)](#)
- [ST\\_M \(p. 1242\)](#)
- [ST\\_MakeEnvelope \(p. 1243\)](#)
- [ST\\_MakeLine \(p. 1244\)](#)
- [ST\\_MakePoint \(p. 1245\)](#)
- [ST\\_MakePolygon \(p. 1246\)](#)
- [ST\\_MemSize \(p. 1248\)](#)
- [ST\\_MMax \(p. 1248\)](#)
- [ST\\_MMin \(p. 1249\)](#)
- [ST\\_Multi \(p. 1250\)](#)
- [ST\\_NDims \(p. 1251\)](#)
- [ST\\_NPoints \(p. 1252\)](#)
- [ST\\_NRings \(p. 1253\)](#)
- [ST\\_NumGeometries \(p. 1254\)](#)
- [ST\\_NumInteriorRings \(p. 1255\)](#)
- [ST\\_NumPoints \(p. 1255\)](#)
- [ST\\_Perimeter \(p. 1256\)](#)
- [ST\\_Perimeter2D \(p. 1257\)](#)
- [ST\\_Point \(p. 1257\)](#)
- [ST\\_PointN \(p. 1258\)](#)
- [ST\\_Points \(p. 1259\)](#)
- [ST\\_Polygon \(p. 1260\)](#)
- [ST\\_RemovePoint \(p. 1261\)](#)
- [ST\\_Reverse \(p. 1261\)](#)
- [ST\\_SetPoint \(p. 1262\)](#)
- [ST\\_SetSRID \(p. 1263\)](#)
- [ST\\_Simplify \(p. 1264\)](#)
- [ST\\_SRID \(p. 1265\)](#)
- [ST\\_StartPoint \(p. 1266\)](#)
- [ST\\_Touches \(p. 1266\)](#)
- [ST\\_Transform \(p. 1267\)](#)
- [ST\\_Union \(p. 1268\)](#)
- [ST\\_Within \(p. 1269\)](#)
- [ST\\_X \(p. 1270\)](#)
- [ST\\_XMax \(p. 1271\)](#)

- [ST\\_XMin \(p. 1271\)](#)
- [ST\\_Y \(p. 1272\)](#)
- [ST\\_YMax \(p. 1272\)](#)
- [ST\\_YMin \(p. 1273\)](#)
- [ST\\_Z \(p. 1274\)](#)
- [ST\\_ZMax \(p. 1275\)](#)
- [ST\\_ZMin \(p. 1275\)](#)
- [SupportsBBox \(p. 1276\)](#)

## AddBBox

AddBBox returns a copy of the input geometry that supports encoding with a precomputed bounding box. For more information about support for bounding boxes, see [Bounding box \(p. 229\)](#).

### Syntax

```
AddBBox(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

GEOMETRY

If *geom* is null, then null is returned.

### Examples

The following SQL returns a copy of an input polygon geometry that supports being encoded with a bounding box.

```
SELECT ST_AsText(AddBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0)')));
```

```
st_astext

POLYGON((0 0,1 0,0 1,0 0))
```

## DropBBox

DropBBox returns a copy of the input geometry that doesn't support encoding with a precomputed bounding box. For more information about support for bounding boxes, see [Bounding box \(p. 229\)](#).

### Syntax

```
DropBBox(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

GEOMETRY

If *geom* is null, then null is returned.

## Examples

The following SQL returns a copy of an input polygon geometry that doesn't support being encoded with a bounding box.

```
SELECT ST_AsText(DropBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0)')));
```

```
st_astext

POLYGON((0 0,1 0,0 1,0 0))
```

## GeometryType

GeometryType returns the subtype of an input geometry as a string.

## Syntax

```
GeometryType(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

VARCHAR representing the subtype of *geom*.

If *geom* is null, then null is returned.

The values returned are as follows.

| Returned string value for 2D, 3DZ, 4D geometries | Returned string value for 3DM geometries | Geometry subtype                                |
|--------------------------------------------------|------------------------------------------|-------------------------------------------------|
| POINT                                            | POINTM                                   | Returned if <i>geom</i> is a POINT subtype      |
| LINESTRING                                       | LINESTRINGM                              | Returned if <i>geom</i> is a LINESTRING subtype |

| Returned string value for 2D, 3DZ, 4D geometries | Returned string value for 3DM geometries | Geometry subtype                                        |
|--------------------------------------------------|------------------------------------------|---------------------------------------------------------|
| POLYGON                                          | POLYGONM                                 | Returned if <i>geom</i> is a POLYGON subtype            |
| MULTIPOINT                                       | MULTIPOINTM                              | Returned if <i>geom</i> is a MULTIPOINT subtype         |
| MULTILINESTRING                                  | MULTILINESTRINGM                         | Returned if <i>geom</i> is a MULTILINESTRING subtype    |
| MULTIPOLYGON                                     | MULTIPOLYGONM                            | Returned if <i>geom</i> is a MULTIPOLYGON subtype       |
| GEOMETRYCOLLECTION                               | GEOMETRYCOLLECTIONM                      | Returned if <i>geom</i> is a GEOMETRYCOLLECTION subtype |

## Examples

The following SQL converts a well-known text (WKT) representation of a polygon and returns the GEOMETRY subtype as a string.

```
SELECT GeometryType(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
geometrytype

POLYGON
```

## ST\_AddPoint

ST\_AddPoint returns a linestring geometry that is the same as the input geometry with a point added. If an index is provided, then the point is added at the index position. If the index is -1 or not provided, then the point is appended to the linestring.

The index is zero-based. The spatial reference system identifier (SRID) of the result is the same as that of the input geometry.

The dimension of the returned geometry is the same as that of the *geom1* value. If *geom1* and *geom2* have different dimensions, *geom2* is projected to the dimension of *geom1*.

### Syntax

```
ST_AddPoint(geom1, geom2)
```

```
ST_AddPoint(geom1, geom2, index)
```

### Arguments

#### *geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be LINESTRING.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be POINT. The point can be the empty point.

*index*

A value of data type INTEGER that represents the position of a zero-based index.

## Return type

GEOMETRY

If *geom1*, *geom2*, or *index* is null, then null is returned.

If *geom2* is the empty point, then a copy of *geom1* is returned.

If *geom1* is not a LINESTRING, then an error is returned.

If *geom2* is not a POINT, then an error is returned.

If *index* is out of range, then an error is returned. Valid values for the index position are -1 or a value between 0 and ST\_NumPoints(*geom1*).

## Examples

The following SQL adds a point to a linestring to make it a closed linestring.

```
WITH tmp(g) AS (SELECT ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326))
SELECT ST_AsEWKT(ST_AddPoint(g, ST_StartPoint(g))) FROM tmp;
```

```
st_asewkt

SRID=4326;LINESTRING(0 0,10 0,10 10,5 5,0 5,0 0)
```

The following SQL adds a point to a specific position in a linestring.

```
WITH tmp(g) AS (SELECT ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326))
SELECT ST_AsEWKT(ST_AddPoint(g, ST_SetSRID(ST_Point(5, 10), 4326), 3)) FROM tmp;
```

```
st_asewkt

SRID=4326;LINESTRING(0 0,10 0,10 10,5 10,5 5,0 5)
```

## ST\_Angle

ST\_Angle returns the angle in radians between points measured clockwise as follows:

- If three points are input, then the returned angle P1-P2-P3 is measured as if the angle was obtained by rotating from P1 to P3 around P2 clockwise.
- If four points are input, then the returned clockwise angle formed by the directed lines P1-P2 and P3-P4 is returned. If the input is a degenerate case (that is, P1 equals P2, or P3 equals P4), then null is returned.

The return value is in radians and in the range  $[0, 2\pi]$ .

ST\_Angle operates on 2D projections of the input geometries.

## Syntax

```
ST_Angle(geom1, geom2, geom3)
```

```
ST_Angle(geom1, geom2, geom3, geom4)
```

## Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be POINT.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be POINT.

*geom3*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be POINT.

*geom4*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be POINT.

## Return type

DOUBLE PRECISION.

If *geom1* equals *geom2*, or *geom2* equals *geom3*, then a null is returned.

If *geom1*, *geom2*, *geom3*, or *geom4* is null, then a null is returned.

If any of *geom1*, *geom2*, *geom3*, or *geom4* is the empty point, then an error is returned.

If *geom1*, *geom2*, *geom3*, and *geom4* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

## Examples

The following SQL returns the angle converted to degrees of three input points.

```
SELECT ST_Angle(ST_Point(1,1), ST_Point(0,0), ST_Point(1,0)) / Pi() * 180.0 AS angle;
```

```
angle
```

```

```

```
45
```

The following SQL returns the angle converted to degrees of four input points.

```
SELECT ST_Angle(ST_Point(1,1), ST_Point(0,0), ST_Point(1,0), ST_Point(2,0)) / Pi() * 180.0
AS angle;
```

```
angle

225
```

## ST\_Area

For an input geometry, ST\_Area returns the Cartesian area of the 2D projection. The area units are the same as the units in which the coordinates of the input geometry are expressed. For points, linestrings, multipoints, and multilinestrings, the function returns 0. For geometry collections, it returns the sum of the areas of the geometries in the collection.

For an input geography, ST\_Area returns the geodesic area of the 2D projection of an input areal geography computed on the spheroid determined by the SRID. The unit of length is in square meters. The function returns zero (0) for points, multipoints, and linear geographies. When the input is a geometry collection, the function returns the sum of the areas of the areal geographies in the collection.

### Syntax

```
ST_Area(geo)
```

### Arguments

*geo*

A value of data type GEOMETRY or GEOGRAPHY, or an expression that evaluates to a GEOMETRY or GEOGRAPHY type.

### Return type

DOUBLE PRECISION

If *geo* is null, then null is returned.

### Examples

The following SQL returns the Cartesian area of a multipolygon.

```
SELECT ST_Area(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,10,0 0),(10 0,20 0,20 10,10 0)))'));
```

```
st_area

100
```

The following SQL returns the area of a polygon in a geography.

```
SELECT ST_Area(ST_GeogFromText('polygon((34 35, 28 30, 25 34, 34 35))'));
```

st\_area  
-----  
201824655743.383

The following SQL returns zero for a linear geography.

```
SELECT ST_Area(ST_GeogFromText('multipoint(0 0, 1 1, -21.32 121.2)'));
```

st\_area  
-----  
0

## ST\_AsBinary

`ST_AsBinary` returns the hexadecimal well-known binary (WKB) representation of an input geometry. For 3DZ, 3DM, and 4D geometries, `ST_AsBinary` uses the Open Geospatial Consortium (OGC) standard value for the geometry type.

## Syntax

`ST_AsBinary(geom)`

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

## VARBYTF

If *geom* is null, then null is returned.

## Examples

The following SQL returns the hexadecimal WKB representation of a polygon.

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))', 4326));
```

st\_asbinary

---

ST\_AsEWKB

`ST_AsEWKB` returns the extended well-known binary (EWKB) representation of an input geometry. For 3DZ, 3DM, and 4D geometries, `ST_AsEWKB` uses the Open Geospatial Consortium (OGC) standard value for the geometry type.

## Syntax

`ST_AsEWKB(geom)`

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

VARBYTE

If *geom* is null, then null is returned.

## Examples

The following SQL returns the hexadecimal EWKB representation of a polygon.

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

st\_asewkb

0103000020E610000001000000050000

## ST\_AsEWKT

`ST_AsEWKT` returns the extended well-known text (EWKT) representation of an input geometry or geography. For 3DZ, 3DM, and 4D geometries, `ST_AsEWKT` appends Z, M, or ZM to the WKT value for the geometry type.

## Syntax

`ST_AsEWKT(geo)`

`ST_AsEWKT(geo, precision)`

## Arguments

geo

A value of data type GEOMETRY or GEOGRAPHY, or an expression that evaluates to a GEOMETRY or GEOGRAPHY type.

### *precision*

A value of data type INTEGER. For geometries, the coordinates of *geo* are displayed using the specified precision 1–20. If *precision* is not specified, the default is 15. For geographies, the coordinates of *geo* are displayed using the specified precision. If *precision* is not specified, the default is 15.

## Return type

VARCHAR

If *geo* is null, then null is returned.

If *precision* is null, then null is returned.

If the result is larger than a 64-KB VARCHAR, then an error is returned.

## Examples

The following SQL returns the EWKT representation of a linestring.

```
SELECT ST_AsEWKT(ST_GeomFromText('LINESTRING(3.141592653589793
-6.283185307179586,2.718281828459045 -1.414213562373095)', 4326));
```

```
st_asewkt

SRID=4326;LINESTRING(3.14159265358979 -6.28318530717959,2.71828182845905
-1.41421356237309)
```

The following SQL returns the EWKT representation of a linestring. The coordinates of the geometries are displayed with six digits of precision.

```
SELECT ST_AsEWKT(ST_GeomFromText('LINESTRING(3.141592653589793
-6.283185307179586,2.718281828459045 -1.414213562373095)', 4326), 6);
```

```
st_asewkt

SRID=4326;LINESTRING(3.14159 -6.28319,2.71828 -1.41421)
```

The following SQL returns the EWKT representation of a geography.

```
SELECT ST_AsEWKT(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
st_asewkt

SRID=4326;LINESTRING(110 40,2 3,-10 80,-7 9)
```

## ST\_AsGeoJSON

`ST_AsGeoJSON` returns the GeoJSON representation of an input geometry or geography. For more information about GeoJSON, see [GeoJSON](#) in Wikipedia.

For 3DZ and 4D geometries, the output geometry is a 3DZ projection of the input 3DZ or 4D geometry. That is, the x, y, and z coordinates are present in the output. For 3DM geometries, the output geometry is a 2D projection of the input 3DM geometry. That is, only x and y coordinates are present in the output.

For input geographies, `ST_AsGeoJSON` returns the GeoJSON representation of an input geography. The coordinates of the geography are displayed using the specified precision.

## Syntax

```
ST_AsGeoJSON(geo)
```

```
ST_AsGeoJSON(geo, precision)
```

## Arguments

*geo*

A value of data type GEOMETRY or GEOGRAPHY, or an expression that evaluates to a GEOMETRY or GEOGRAPHY type.

*precision*

A value of data type INTEGER. For geometries, the coordinates of *geo* are displayed using the specified precision 1–20. If *precision* is not specified, the default is 15. For geographies, the coordinates of *geo* are displayed using the specified precision. If *precision* is not specified, the default is 15.

## Return type

VARCHAR

If *geo* is null, then null is returned.

If *precision* is null, then null is returned.

If the result is larger than a 64-KB VARCHAR, then an error is returned.

## Examples

The following SQL returns the GeoJSON representation of a linestring.

```
SELECT ST_AsGeoJSON(ST_GeomFromText('LINESTRING(3.141592653589793
-6.283185307179586,2.718281828459045 -1.414213562373095)'));
```

```
st_asgeojson

{"type": "LineString", "coordinates": [[3.14159265358979, -6.28318530717959],
[2.71828182845905, -1.41421356237309]]}
```

The following SQL returns the GeoJSON representation of a linestring. The coordinates of the geometries are displayed with six digits of precision.

```
SELECT ST_AsGeoJSON(ST_GeomFromText('LINESTRING(3.141592653589793
-6.283185307179586,2.718281828459045 -1.414213562373095)'), 6);
```

```
st_asgeojson

{"type": "LineString", "coordinates": [[3.14159, -6.28319], [2.71828, -1.41421]]}}
```

The following SQL returns the GeoJSON representation of a geography.

```
SELECT ST_AsGeoJSON(ST_GeogFromText('LINESTRING(110 40, 23, -10 80, -7 9)'));
```

`st_asgeojson`

```
{"type": "LineString", "coordinates": [[110, 40], [2, 3], [-10, 80], [-7, 9]]}
```

## ST\_AsHexWKB

`ST_AsHexWKB` returns the hexadecimal well-known binary (WKB) representation of an input geometry or geography using ASCII hexadecimal characters (0–9, A–F). For 3DZ, 3DM, and 4D geometries or geographies, `ST_AsHexWKB` uses the Open Geospatial Consortium (OGC) standard value for the geometry or geography type.

## Syntax

`ST_AsHexWKB(geo)`

## Arguments

geo

A value of data type GEOMETRY or GEOGRAPHY, or an expression that evaluates to a GEOMETRY or GEOGRAPHY type.

## Return type

**VARCHAR**

If *geo* is null, then null is returned.

If the result is larger than a 64-KB VARCHAR, then an error is returned.

## Examples

The following SQL returns the hexadecimal WKB representation of a polygon in a geometry.

```
SELECT ST_AsHexWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

st ashexwkb

-----

The following SQL returns the hexadecimal WKB representation of a polygon in a geography.

```
SELECT ST_AsHexWKB(ST_GeogFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))'));
```

st ashexwkb

-----

## ST\_AsHexEWKB

`ST_AsHexEWKB` returns the extended well-known binary (EWKB) representation of an input geometry or geography using ASCII hexadecimal characters (0–9, A–F). For 3DZ, 3DM, and 4D geometries or geographies, `ST_AsHexEWKB` uses the PostGIS extended WKB value for the geometry or geography type.

## Syntax

ST\_AsHexEWKB(*geo*)

## Arguments

geo

A value of data type GEOMETRY or GEOGRAPHY, or an expression that evaluates to a GEOMETRY or GEOGRAPHY type.

## Return type

VARCHAR

If *geo* is null, then null is returned.

If the result is larger than a 64-KB VARCHAR, then an error is returned.

## Examples

The following SQL returns the hexadecimal EWKB representation of a polygon in a geometry.

```
SELECT ST_AsHexEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

st ashexewkb

0103000020E610000001000000050000

The following SQL returns the hexadecimal EWKB representation of a polygon in a geography.

```
SELECT ST_AsHexEWKB(ST_GeogFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))'));
```

st ashexewkh

0103000030E61000000010000000050000

## ST\_ASTEXT

`ST_AsText` returns the well-known text (WKT) representation of an input geometry or geography. For 3DZ, 3DM, and 4D geometries or geographies, `ST_AsEWKT` appends Z, M, or ZM to the WKT value for the geometry or geography type.

## Syntax

`ST_AsText(geo)`

```
ST_AsText(geo, precision)
```

## Arguments

*geo*

A value of data type GEOMETRY or GEOGRAPHY, or an expression that evaluates to a GEOMETRY or GEOGRAPHY type.

*precision*

A value of data type INTEGER. For geometries, the coordinates of *geo* are displayed using the specified precision 1–20. If *precision* is not specified, the default is 15. For geographies, the coordinates of *geo* are displayed using the specified precision. If *precision* is not specified, the default is 15.

## Return type

VARCHAR

If *geo* is null, then null is returned.

If *precision* is null, then null is returned.

If the result is larger than a 64-KB VARCHAR, then an error is returned.

## Examples

The following SQL returns the WKT representation of a linestring.

```
SELECT ST_AsText(ST_GeomFromText('LINESTRING(3.141592653589793
-6.283185307179586,2.718281828459045 -1.414213562373095)', 4326));
```

```
st_astext

LINESTRING(3.14159265358979 -6.28318530717959,2.71828182845905 -1.41421356237309)
```

The following SQL returns the WKT representation of a linestring. The coordinates of the geometries are displayed with six digits of precision.

```
SELECT ST_AsText(ST_GeomFromText('LINESTRING(3.141592653589793
-6.283185307179586,2.718281828459045 -1.414213562373095)', 4326), 6);
```

```
st_astext

LINESTRING(3.14159 -6.28319,2.71828 -1.41421)
```

The following SQL returns the WKT representation of a geography.

```
SELECT ST_AsText(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
st_astext

LINESTRING(110 40,2 3,-10 80,-7 9)
```

## ST\_Azimuth

ST\_Azimuth returns the north-based Cartesian azimuth using the 2D projections of the two input points.

### Syntax

```
ST_Azimuth(point1, point2)
```

### Arguments

*point1*

A POINT value of data type GEOMETRY. The spatial reference system identifier (SRID) of *point1* must match the SRID of *point2*.

*point2*

A POINT value of data type GEOMETRY. The SRID of *point2* must match the SRID of *point1*.

### Return type

A number that is an angle in radians of DOUBLE PRECISION data type. Values range from 0 (inclusive) to 2 pi (exclusive).

If *point1* or *point2* is the empty point, then an error is returned.

If either *point1* or *point2* is null, then null is returned.

If *point1* and *point2* are equal, then null is returned.

If *point1* or *point2* is not a point, then an error is returned.

If *point1* and *point2* don't have the value for the spatial reference system identifier (SRID), then an error is returned.

### Examples

The following SQL returns the azimuth of the input points.

```
SELECT ST_Azimuth(ST_Point(1,2), ST_Point(5,6));
```

```
st_azimuth

0.7853981633974483
```

## ST\_Boundary

ST\_Boundary returns the boundary of an input geometry as follows:

- If the input geometry is empty (that is, it contains no points) it is returned as is.

- If the input geometry is a point or nonempty multipoint, an empty geometry collection is returned.
- If the input is a linestring or a multilinestring, then a multipoint containing all the points on the boundary is returned. The multipoint might be empty).
- If the input is a polygon that does not have any interior rings, then a closed linestring representing its boundary is returned.
- If the input is a polygon that has interior rings, or is a multipolygon, then a multilinestring is returned. The multilinestring contains all the boundaries of all the rings in the areal geometry as closed linestrings.

To determine point equality, ST\_Boundary operates on the 2D projection of the input geometry. If the input geometry is empty, a copy of it is returned in the same dimension as the input. For nonempty 3DM and 4D geometries, their m coordinates are dropped. In the special case of 3DZ and 4D multilinestrings, the z coordinates of the multilinestring's boundary points are computed as the averages of the distinct z-values of the linestring boundary points with the same 2D projection.

## Syntax

```
ST_Boundary(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

GEOMETRY

If *geom* is null, then null is returned.

If *geom* is a GEOMETRYCOLLECTION, then an error is returned.

## Examples

The following SQL returns the boundary of the input polygon as a multilinestring.

```
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(1 1,1 2,2 1,1 1))')));
```

```
st_asewkt

MULTILINESTRING((0 0,10 0,10 10,0 10,0 0),(1 1,1 2,2 1,1 1))
```

## ST\_Buffer

ST\_Buffer returns 2D geometry that represents all points whose distance from the input geometry projected on the xy-Cartesian plane is less than or equal to the input distance.

## Syntax

```
ST_Buffer(geom, distance)
```

```
ST_Buffer(geom, distance, number_of_segments_per_quarter_circle)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*distance*

A value of data type DOUBLE PRECISION that represents distance (or radius) of the buffer.

*number\_of\_segments\_per\_quarter\_circle*

A value of data type INTEGER. This value determines the number of points to approximate a quarter circle around each vertex of the input geometry. Negative values default to zero. The default is 8.

## Return type

GEOMETRY

The ST\_Buffer function returns two-dimensional (2D) geometry in the xy-Cartesian plane.

If *geom* is a GEOMETRYCOLLECTION, then an error is returned.

## Examples

The following SQL returns the buffer of the input linestring.

```
SELECT ST_AsEWKT(ST_Buffer(ST_GeomFromText('LINESTRING(1 2,5 2,5 8)'), 2));
```

```
st_asewkt
POLYGON((-1 2,-0.96157056080646 2.39018064403226,-0.847759065022573
2.76536686473018,-0.662939224605089 3.11114046603921,-0.414213562373093
3.4142135623731,-0.111140466039201 3.66293922460509,0.234633135269824
3.84775906502257,0.609819355967748 3.96157056080646,1 4,3 4,3 8,3.03842943919354
8.39018064403226,3.15224093497743 8.76536686473018,3.33706077539491
9.11114046603921,3.58578643762691 9.4142135623731,3.8888595339608
9.66293922460509,4.23463313526982 9.84775906502257,4.60981935596775 9.96157056080646,5
10,5.39018064403226 9.96157056080646,5.76536686473018 9.84775906502257,6.11114046603921
9.66293922460509,6.4142135623731 9.41421356237309,6.66293922460509
9.1111404660392,6.84775906502258 8.76536686473017,6.96157056080646 8.39018064403225,7 8,7
2,6.96157056080646 1.60981935596774,6.84775906502257 1.23463313526982,6.66293922460509
0.888859533960796,6.41421356237309 0.585786437626905,6.1111404660392
0.33706077539491,5.76536686473018 0.152240934977427,5.39018064403226
0.0384294391935391,5 0,1 0,0.609819355967744 0.0384294391935391,0.234633135269821
0.152240934977427,-0.111140466039204 0.337060775394909,-0.414213562373095
0.585786437626905,-0.662939224605091 0.888859533960796,-0.847759065022574
1.23463313526982,-0.961570560806461 1.60981935596774,-1 2))
```

The following SQL returns the buffer of the input point geometry which approximates a circle. Because the command doesn't specify the number of segments per quarter circle, the function uses the default value of eight segments to approximate the quarter circle.

```
SELECT ST_AsEWKT(ST_Buffer(ST_GeomFromText('POINT(3 4)'), 2));
```

```
st_asewkt
POLYGON((1 4,1.03842943919354 4.39018064403226,1.15224093497743
4.76536686473018,1.33706077539491 5.11114046603921,1.58578643762691
```

```

5.4142135623731,1.8888595339608 5.66293922460509,2.23463313526982
5.84775906502257,2.60981935596775 5.96157056080646,3 6,3.39018064403226
5.96157056080646,3.76536686473019 5.84775906502257,4.11114046603921
5.66293922460509,4.4142135623731 5.41421356237309,4.66293922460509
5.1111404660392,4.84775906502258 4.76536686473017,4.96157056080646 4.39018064403225,5
4,4.96157056080646 3.60981935596774,4.84775906502257 3.23463313526982,4.66293922460509
2.8888595339608,4.41421356237309 2.58578643762691,4.1111404660392
2.33706077539491,3.76536686473018 2.15224093497743,3.39018064403226 2.03842943919354,3
2,2.60981935596774 2.03842943919354,2.23463313526982 2.15224093497743,1.8888595339608
2.33706077539491,1.58578643762691 2.58578643762691,1.33706077539491
2.8888595339608,1.15224093497743 3.23463313526982,1.03842943919354 3.60981935596774,1 4)

```

The following SQL returns the buffer of the input point geometry which approximates a circle. Because the command specifies 3 as the number of segments per quarter circle, the function uses three segments to approximate the quarter circle.

```
SELECT ST_AsEWKT(ST_Buffer(ST_GeomFromText('POINT(3 4)'), 2, 3));
```

```

st_asewkt
POLYGON((1 4,1.26794919243112 5,2 5.73205080756888,3 6,4 5.73205080756888,4.73205080756888
5,5 4,4.73205080756888 3,4 2.26794919243112,3 2,2 2.26794919243112,1.26794919243112 3,1
4))

```

## ST\_Centroid

`ST_Centroid` returns a point that represents a centroid of a geometry as follows:

- For `POINT` geometries, it returns the point whose coordinates are the average of the coordinates of the points in the geometry.
- For `LINESTRING` geometries, it returns the point whose coordinates are the weighted average of the midpoints of the segments of the geometry, where the weights are the lengths of the segments of the geometry.
- For `POLYGON` geometries, it returns the point whose coordinates are the weighted average of the centroids of a triangulation of the areal geometry where the weights are the areas of the triangles in the triangulation.
- For geometry collections, it returns the weighted average of the centroids of the geometries of maximum topological dimension in the geometry collection.

## Syntax

```
ST_Centroid(geom)
```

## Arguments

*geom*

A value of data type `GEOMETRY` or an expression that evaluates to a `GEOMETRY` type.

## Return type

`GEOMETRY`

If *geom* is null, then null is returned.

If *geom* is empty, then null is returned.

## Examples

The following SQL returns central point of an input linestring.

```
SELECT ST_AsEWKT(ST_Centroid(ST_GeomFromText('LINESTRING(110 40, 2 3, -10 80, -7 9, -22
-33)', 4326)))
```

```
st_asewkt
```

```

SRID=4326;POINT(15.6965103455214 27.0206782881905)
```

## ST\_Collect

`ST_Collect` has two variants. One accepts two geometries, and one accepts an aggregate expression.

The first variant of `ST_Collect` creates a geometry from the input geometries. The order of the input geometries is preserved. This variant works as follows:

- If both input geometries are points, then a `MULTIPOINT` with two points is returned.
- If both input geometries are linestrings, then a `MULTILINESTRING` with two linestrings is returned.
- If both input geometries are polygons, then a `MULTIPOLYGON` with two polygons is returned.
- Otherwise, a `GEOMETRYCOLLECTION` with two input geometries is returned.

The second variant of `ST_Collect` creates a geometry from geometries in a geometry column. There isn't a determined return order of the geometries. Specify the `WITHIN GROUP (ORDER BY ...)` clause to specify the order of the returned geometries. This variant works as follows:

- If all non-NULL rows in the input aggregate expression are points, then a multipoint containing all the points in the aggregate expression is returned.
- If all non-NULL rows in the aggregate expression are linestrings, then a multilinestring containing all the linestrings in the aggregate expression is returned.
- If all non-NULL rows in the aggregate expression are polygons, the result is a multipolygon containing all the polygons in the aggregate expression is returned.
- Otherwise, a `GEOMETRYCOLLECTION` containing all the geometries in the aggregate expression is returned.

The `ST_Collect` returns the geometry of the same dimension as the input geometries. All input geometries must be of the same dimension.

## Syntax

```
ST_Collect(geom1, geom2)
```

```
ST_Collect(aggregate_expression) [WITHIN GROUP (ORDER BY sort_expression1 [ASC | DESC]
[, sort_expression2 [ASC | DESC] ...])]
```

## Arguments

*geom1*

A value of data type `GEOMETRY` or an expression that evaluates to a `GEOMETRY` type.

### *geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### *aggregate\_expression*

A column of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

[WITHIN GROUP (ORDER BY *sort\_expression1* [ASC | DESC] [, *sort\_expression2* [ASC | DESC] ...])]

An optional clause that specifies the sort order of the aggregated values. The ORDER BY clause contains a list of sort expressions. Sort expressions are expressions similar to valid sort expressions in a query select list, such as a column name. You can specify ascending (ASC) or descending (DESC) order. The default is ASC.

## Return type

GEOMETRY of subtype MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, or GEOMETRYCOLLECTION.

The spatial reference system identifier (SRID) value of the returned geometry is the SRID value of the input geometries.

If both *geom1* or *geom2* are null, then null is returned.

If all rows of *aggregate\_expression* are null, then null is returned.

If *geom1* is null, then a copy of *geom2* is returned. Likewise, if *geom2* is null, then a copy of *geom1* is returned.

If *geom1* and *geom2* have different SRID values, then an error is returned.

If two geometries in *aggregate\_expression* have different SRID values, then an error is returned.

If the returned geometry is larger than the maximum size of a GEOMETRY, then an error is returned.

If *geom1* and *geom2* are of different dimensions, then an error is returned.

If two geometries in *aggregate\_expression* are of different dimensions, then an error is returned.

## Examples

The following SQL returns a geometry collection that contains the two input geometries.

```
SELECT ST_AsText(ST_Collect(ST_GeomFromText('LINESTRING(0 0,1 1)'),
 ST_GeomFromText('POLYGON((10 10,20 10,20,10 10))')));
```

```
st_astext

GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),POLYGON((10 10,20 10,20,10 10)))
```

The following SQL collects all the geometries from a table into a geometry collection.

```
WITH tbl(g) AS (SELECT ST_GeomFromText('POINT(1 2)', 4326) UNION ALL
 SELECT ST_GeomFromText('LINESTRING(0 0,10 0)', 4326) UNION ALL
 SELECT ST_GeomFromText('MULTIPOINT(13 4,8 5,4 4)', 4326) UNION ALL
 SELECT NULL::geometry UNION ALL
```

```
SELECT ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))', 4326))
SELECT ST_AsEWKT(ST_Collect(g)) FROM tbl;
```

```
st_astext

SRID=4326;GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(0 0,10 0),MULTIPOINT((13 4),(8 5),(4 4)),POLYGON((0 0,10 0,0 10,0 0)))
```

The following SQL collects all geometries in the table grouped by the `id` column and ordered by this ID. In this example, resulting geometries are grouped by ID as follows:

- id 1 – points in a multipoint.
- id 2 – linestrings in a multilinestring.
- id 3 – mixed subtypes in a geometry collection.
- id 4 – polygons in a multipolygon.
- id 5 – null and the result is null.

```
WITH tbl(id, g) AS (SELECT 1, ST_GeomFromText('POINT(1 2)', 4326) UNION ALL
SELECT 1, ST_GeomFromText('POINT(4 5)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(0 0,10 0)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(10 0,20 -5)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTIPOINT(13 4,8 5,4 4)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTILINESTRING((-1 -1,-2 -2),(-3 -3,-5 -5))', 4326) UNION ALL
SELECT 4, ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))', 4326) UNION ALL
SELECT 4, ST_GeomFromText('POLYGON((20 20,20 30,30 20,20 20))', 4326) UNION ALL
SELECT 1, NULL::geometry UNION ALL SELECT 2, NULL::geometry UNION ALL
SELECT 5, NULL::geometry UNION ALL SELECT 5, NULL::geometry)
SELECT id, ST_AsEWKT(ST_Collect(g)) FROM tbl GROUP BY id ORDER BY id;
```

| id                                                                                                         | st_asewkt |
|------------------------------------------------------------------------------------------------------------|-----------|
| -----                                                                                                      |           |
| +-----                                                                                                     |           |
| 1   SRID=4326;MULTIPOINT((1 2),(4 5))                                                                      |           |
| 2   SRID=4326;MULTILINESTRING((0 0,10 0),(10 0,20 -5))                                                     |           |
| 3   SRID=4326;GEOMETRYCOLLECTION(MULTIPOINT((13 4),(8 5),(4 4)),MULTILINESTRING((-1 -1,-2),(-3 -3,-5 -5))) |           |
| 4   SRID=4326;MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((20 20,20 30,30 20,20 20)))                              |           |
| 5                                                                                                          |           |

The following SQL collects all geometries from a table in a geometry collection. Results are ordered in descending order by `id`, and then lexicographically based on their minimum and maximum x-coordinates.

```
WITH tbl(id, g) AS (
SELECT 1, ST_GeomFromText('POINT(4 5)', 4326) UNION ALL
SELECT 1, ST_GeomFromText('POINT(1 2)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(10 0,20 -5)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(0 0,10 0)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTIPOINT(13 4,8 5,4 4)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTILINESTRING((-1 -1,-2 -2),(-3 -3,-5 -5))', 4326) UNION ALL
SELECT 4, ST_GeomFromText('POLYGON((20 20,20 30,30 20,20 20))', 4326) UNION ALL
SELECT 4, ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))', 4326) UNION ALL
SELECT 1, NULL::geometry UNION ALL SELECT 2, NULL::geometry UNION ALL
SELECT 5, NULL::geometry UNION ALL SELECT 5, NULL::geometry)
```

```
SELECT ST_AsEWKT(ST_Collect(g) WITHIN GROUP (ORDER BY id DESC, ST_XMin(g), ST_XMax(g)))
FROM tbl;
```

```
st_asewkt
```

```
SRID=4326;GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0 10,0 0)),POLYGON((20 20,20 30,30
20,20 20)),MULTILINESTRING((-1 -1,-2 -2),(-3 -3,-5 -5)),MULTIPOINT((13 4),(8 5),(4
4)),LINESTRING(0 0,10 0),LINESTRING(10 0,20 -5),POINT(1 2),POINT(4 5))
```

## ST\_Contains

`ST_Contains` returns true if the 2D projection of the first input geometry contains the 2D projection of the second input geometry. Geometry A contains geometry B if every point in B is a point in A, and their interiors have nonempty intersection.

`ST_Contains(A, B)` is equivalent to `ST_Within(B, A)`.

### Syntax

```
ST_Contains(geom1, geom2)
```

### Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. This value is compared with *geom1* to determine if it is contained within *geom1*.

### Return type

BOOLEAN

If *geom1* or *geom2* is null, then null is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geom1* or *geom2* is a geometry collection, then an error is returned.

### Examples

The following SQL checks if the first polygon contains the second polygon.

```
SELECT ST_Contains(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_contains
```

false

## ST\_ContainsProperly

`ST_ContainsProperly` returns true if both input geometries are nonempty, and all points of the 2D projection of the second geometry are interior points of the 2D projection of the first geometry.

### Syntax

```
ST_ContainsProperly(geom1, geom2)
```

### Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype can't be GEOMETRYCOLLECTION.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype can't be GEOMETRYCOLLECTION. This value is compared with *geom1* to determine if all its points are interior points of *geom1*.

### Return type

BOOLEAN

If *geom1* or *geom2* is null, then null is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geom1* or *geom2* is a geometry collection, then an error is returned.

### Examples

The following SQL returns the values of `ST_Contains` and `ST_ContainsProperly` where the input linestring intersects the interior and the boundary of the input polygon (but not its exterior). The polygon contains the linestring but doesn't properly contain the linestring.

```
WITH tmp(g1, g2)
AS (SELECT ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0))'),
 ST_GeomFromText('LINESTRING(5 5,10 5,10 6,5 5)')) SELECT ST_Contains(g1, g2),
 ST_ContainsProperly(g1, g2)
FROM tmp;
```

| st_contains | st_containsproperly |
|-------------|---------------------|
| t           | f                   |

## ST\_ConvexHull

`ST_ConvexHull` returns a geometry that represents the convex hull of the nonempty points contained in the input geometry.

For empty input, the resulting geometry is the same as the input geometry. For all nonempty input, the function operates on the 2D projection of the input geometry. However, the dimension of the output geometry depends on the dimension of the input geometry. More specifically, when the input geometry is a nonempty 3DM or 3D geometry,  $m$  coordinates are dropped. That is, the dimension of the returned geometry is 2D or 3DZ, respectively. If the input is a nonempty 2D or 3DZ geometry, the resulting geometry has the same dimension.

## Syntax

```
ST_ConvexHull(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

GEOMETRY

The spatial reference system identifier (SRID) value of the returned geometry is the SRID value of the input geometry.

If *geom* is null, then null is returned.

The values returned are as follows.

| Number of points on the convex hull | Geometry subtype                                                                                                                                                                       |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0                                   | A copy of <i>geom</i> is returned.                                                                                                                                                     |
| 1                                   | A POINT subtype is returned.                                                                                                                                                           |
| 2                                   | A LINESTRING subtype is returned. The two points of the returned linestring are lexicographically ordered.                                                                             |
| 3 or greater                        | A POLYGON subtype with no interior rings is returned. The polygon is clockwise oriented, and the first point of the exterior ring is the lexicographically smallest point of the ring. |

## Examples

The following SQL returns the extended well-known text (EWKT) representation of a linestring. In this case, the convex hull returned is a polygon.

```
SELECT ST_AsEWKT(ST_ConvexHull(ST_GeomFromText('LINESTRING(0 0,1 0,0 1,1 1,0.5 0.5)'))) as output;
```

```
output

```

```
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

The following SQL returns the EWKT representation of a linestring. In this case, the convex hull returned is a linestring.

```
SELECT ST_AsEWKT(ST_ConvexHull(ST_GeomFromText('LINESTRING(0 0,1 1,0.2 0.2,0.6 0.6,0.5 0.5)'))) as output;
```

```
output

LINESTRING(0 0,1 1)
```

The following SQL returns the EWKT representation of a multipoint. In this case, the convex hull returned is a point.

```
SELECT ST_AsEWKT(ST_ConvexHull(ST_GeomFromText('MULTIPOINT(0 0,0 0)'))) as output;
```

```
output

POINT(0 0)
```

## ST\_CoveredBy

ST\_CoveredBy returns true if the 2D projection of the first input geometry is covered by the 2D projection of the second input geometry. Geometry A is covered by geometry B if both are nonempty and every point in A is a point in B.

ST\_CoveredBy(A, B) is equivalent to ST\_Covers(B, A).

### Syntax

```
ST_CoveredBy(geom1, geom2)
```

### Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. This value is compared with *geom2* to determine if it's covered by *geom2*.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

BOOLEAN

If *geom1* or *geom2* is null, then null is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geom1* or *geom2* is a geometry collection, then an error is returned.

## Examples

The following SQL checks if the first polygon is covered by the second polygon.

```
SELECT ST_CoveredBy(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_coveredby

true
```

## ST\_Covers

*ST\_Covers* returns true if the 2D projection of the first input geometry covers the 2D projection of the second input geometry. Geometry A covers geometry B if both are nonempty and every point in B is a point in A.

*ST\_Covers(A, B)* is equivalent to *ST\_CoveredBy(B, A)*.

## Syntax

```
ST_Covers(geom1, geom2)
```

## Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. This value is compared with *geom1* to determine if it covers *geom1*.

## Return type

BOOLEAN

If *geom1* or *geom2* is null, then null is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geom1* or *geom2* is a geometry collection, then an error is returned.

## Examples

The following SQL checks if the first polygon covers the second polygon.

```
SELECT ST_Covers(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_covers

false
```

## ST\_Crosses

`ST_Crosses` returns true if the 2D projections of the two input geometries cross each other.

### Syntax

```
ST_Crosses(geom1, geom2)
```

### Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

BOOLEAN

If *geom1* or *geom2* is null, then an error is returned.

If *geom1* or *geom2* is a geometry collection, then an error is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

### Examples

The following SQL checks if the first polygon crosses the second multipoint. In this example, the multipoint intersects both the interior and exterior of the polygon, which is why `ST_Crosses` returns true.

```
SELECT ST_Crosses (ST_GeomFromText('polygon((0 0,10 0,10 10,0 10,0 0))'),
ST_GeomFromText('multipoint(5 5,0 0,-1 -1)'));
```

```
st_crosses

true
```

The following SQL checks if the first polygon crosses the second multipoint. In this example, the multipoint intersects the exterior of the polygon but not its interior, which is why `ST_Crosses` returns false.

```
SELECT ST_Crosses (ST_GeomFromText('polygon((0 0,10 0,10 10,0 10,0 0))'),
ST_GeomFromText('multipoint(0 0,-1 -1)'));
```

```
st_crosses

```

false

## ST\_Dimension

`ST_Dimension` returns the inherent dimension of an input geometry. The *inherent dimension* is the dimension value of the subtype that is defined in the geometry.

For 3DM, 3DZ, and 4D geometry inputs, `ST_Dimension` returns the same result as for 2D geometry inputs.

### Syntax

```
ST_Dimension(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

INTEGER representing the inherent dimension of *geom*.

If *geom* is null, then null is returned.

The values returned are as follows.

| Returned value                                        | Geometry subtype                                                    |
|-------------------------------------------------------|---------------------------------------------------------------------|
| 0                                                     | Returned if <i>geom</i> is a POINT or MULTIPOLYPOINT subtype        |
| 1                                                     | Returned if <i>geom</i> is a LINESTRING or MULTILINESTRING subtype. |
| 2                                                     | Returned if <i>geom</i> is a POLYGON or MULTIPOLYGON subtype        |
| 0                                                     | Returned if <i>geom</i> is an empty GEOMETRYCOLLECTION subtype      |
| Largest dimension of the components of the collection | Returned if <i>geom</i> is a GEOMETRYCOLLECTION subtype             |

### Examples

The following SQL converts a well-known text (WKT) representation of a four-point LINESTRING to a GEOMETRY object and returns the dimension of the linestring.

```
SELECT ST_Dimension(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
```

```
st_dimension

1
```

## ST\_Disjoint

ST\_Disjoint returns true if the 2D projections of the two input geometries have no points in common.

### Syntax

```
ST_Disjoint(geom1, geom2)
```

### Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

BOOLEAN

If *geom1* or *geom2* is null, then null is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geom1* or *geom2* is a geometry collection, then an error is returned.

### Examples

The following SQL checks if the first polygon is disjoint from the second polygon.

```
SELECT ST_Disjoint(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(2 2,2 5,5 5,5 2,2 2))'), ST_Point(4, 4));
```

```
st_disjoint

true
```

## ST\_Distance

For input geometries, ST\_Distance returns the minimum Euclidean distance between the 2D projections of the two input geometry values.

For 3DM, 3DZ, 4D geometries, ST\_Distance returns the Euclidean distance between the 2D projections of two input geometry values.

For input geographies, ST\_Distance returns the geodesic distance of the two 2D points. The unit of distance is in meters. For geographies other than points and empty points an error is returned.

## Syntax

```
ST_Distance(geo1, geo2)
```

## Arguments

*geo1*

A value of data type GEOMETRY or GEOGRAPHY, or an expression that evaluates to a GEOMETRY or GEOGRAPHY type. The data type of *geo1* must be the same as *geo2*.

*geo2*

A value of data type GEOMETRY or GEOGRAPHY, or an expression that evaluates to a GEOMETRY or GEOGRAPHY type. The data type of *geo2* must be the same as *geo1*.

## Return type

DOUBLE PRECISION in the same units as the input geometries or geographies.

If *geo1* or *geo2* is null or empty, then null is returned.

If *geo1* and *geo2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geo1* or *geo2* is a geometry collection, then an error is returned.

## Examples

The following SQL returns the distance between two polygons.

```
SELECT ST_Distance(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
ST_GeomFromText('POLYGON((-1 -3,-2 -1,0 -3,-1 -3))'));
```

```
st_distance

1.4142135623731
```

The following SQL returns the distance between two points in a geography.

```
SELECT ST_Distance(ST_GeogFromText('point(1 1)'),ST_GeogFromText('point(-21.32 121.2)'));
```

```
st_distance

13090291.8796529
```

## ST\_DistanceSphere

ST\_DistanceSphere returns the distance between two point geometries lying on a sphere.

## Syntax

```
ST_DistanceSphere(geom1, geom2)
```

```
ST_DistanceSphere(geom1, geom2, radius)
```

## Arguments

*geom1*

A point value in degrees of data type GEOMETRY lying on a sphere. The first coordinate of the point is the longitude value. The second coordinate of the point is the latitude value. For 3DZ, 3DM, or 4D geometries, only the first two coordinates are used.

*geom2*

A point value in degrees of data type GEOMETRY lying on a sphere. The first coordinate of the point is the longitude value. The second coordinate of the point is the latitude value. For 3DZ, 3DM, or 4D geometries, only the first two coordinates are used.

*radius*

The radius of a sphere of data type DOUBLE PRECISION. If no *radius* is provided, the sphere defaults to Earth and the radius is computed from the World Geodetic System (WGS) 84 representation of the ellipsoid.

## Return type

DOUBLE PRECISION in the same units as the radius.

If *geom1* or *geom2* is null or empty, then null is returned.

If no *radius* is provided, then the result is in meters along the Earth's surface.

If *radius* is a negative number, then an error is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geom1* or *geom2* is not a point, then an error is returned.

## Examples

The following example SQL computes the distances in kilometers between three airport locations in Germany: Berlin Tegel (TXL), Munich International (MUC), and Frankfurt International (FRA).

```
WITH airports_raw(code,lon,lat) AS (
 (SELECT 'MUC', 11.786111, 48.353889) UNION
 (SELECT 'FRA', 8.570556, 50.033333) UNION
 (SELECT 'TXL', 13.287778, 52.559722),
 airports1(code,location) AS (SELECT code, ST_Point(lon, lat) FROM airports_raw),
 airports2(code,location) AS (SELECT * from airports1)
 SELECT (airports1.code || ' <-> ' || airports2.code) AS airports,
 round(ST_DistanceSphere(airports1.location, airports2.location) / 1000, 0) AS
 distance_in_km
 FROM airports1, airports2 WHERE airports1.code < airports2.code ORDER BY 1;
```

| airports    | distance_in_km |
|-------------|----------------|
| FRA <-> MUC | 299            |
| FRA <-> TXL | 432            |

## ST\_DWithin

ST\_DWithin returns true if the Euclidean distance between the 2D projections of the two input geometry values is not larger than a threshold value.

### Syntax

```
ST_DWithin(geom1, geom2, threshold)
```

### Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*threshold*

A value of data type DOUBLE PRECISION. This value is in the units of the input arguments.

### Return type

BOOLEAN

If *geom1* or *geom2* is null, then null is returned.

If *threshold* is negative, then an error is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geom1* or *geom2* is a geometry collection, then an error is returned.

### Examples

The following SQL checks if the distance between two polygons is within five units.

```
SELECT ST_DWithin(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))',
ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'),5);
```

```
st_dwithin

true
```

## ST\_EndPoint

ST\_EndPoint returns the last point of an input linestring. The spatial reference system identifier (SRID) value of the result is the same as that of the input geometry. The dimension of the returned geometry is the same as that of the input geometry.

## Syntax

```
ST_EndPoint(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be LINESTRING.

## Return type

GEOMETRY

If *geom* is null, then null is returned.

If *geom* is empty, then null is returned.

If *geom* isn't a LINESTRING, then null is returned.

## Examples

The following SQL returns an extended well-known text (EWKT) representation of a four-point LINESTRING to a GEOMETRY object and returns the end point of the linestring.

```
SELECT ST_AsEWKT(ST_EndPoint(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326)));

st_asewkt

SRID=4326;POINT(0 5)
```

## ST\_Envelope

ST\_Envelope returns the minimum bounding box of the input geometry, as follows:

- If the input geometry is empty, the returned geometry is a copy of the input geometry.
- If the minimum bounding box of the input geometry degenerates to a point, the returned geometry is a point.
- If the minimum bounding box of the input geometry is one-dimensional, a two-point linestring is returned.
- If none of the preceding is true, the function returns a clockwise-oriented polygon whose vertices are the corners of the minimum bounding box.

The spatial reference system identifier (SRID) of the returned geometry is the same as that of the input geometry.

For all nonempty input, the function operates on the 2D projection of the input geometry.

## Syntax

```
ST_Envelope(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

GEOMETRY

If *geom* is null, then null is returned.

## Examples

The following SQL converts a well-known text (WKT) representation of a four-point LINestring to a GEOMETRY object and returns a polygon whose vertices are the minimum bounding box.

```
SELECT ST_AsText(ST_Envelope(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0,10 0,10,0 0)),LINESTRING(20 10,20 0,10 0))));
```

```
st_astext

POLYGON((0 0,0 10,20 10,20 0,0 0))
```

## ST\_Equals

ST\_Equals returns true if the 2D projections of the input geometries are geometrically equal. Geometries are considered geometrically equal if they have equal point sets and their interiors have a nonempty intersection.

## Syntax

```
ST_Equals(geom1, geom2)
```

## Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. This value is compared with *geom1* to determine if it is equal to *geom1*.

## Return type

BOOLEAN

If *geom1* or *geom2* is null, then an error is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geom1* or *geom2* is a geometry collection, then an error is returned.

## Examples

The following SQL checks if the two polygons are geometrically equal.

```
SELECT ST_Equals(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_equals

false
```

The following SQL checks if the two linestrings are geometrically equal.

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(1 0,10 0)'), ST_GeomFromText('LINESTRING(1 0,5
0,10 0)'));
```

```
st_equals

true
```

## ST\_ExteriorRing

`ST_ExteriorRing` returns a closed linestring that represents the exterior ring of an input polygon. The dimension of the returned geometry is the same as that of the input geometry.

### Syntax

```
ST_ExteriorRing(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

GEOMETRY of subtype LINESTRING.

The spatial reference system identifier (SRID) value of the returned geometry is the SRID value of the input geometry.

If *geom* is null, then null is returned.

If *geom* is not a polygon, then null is returned.

If *geom* is empty, then an empty polygon is returned.

## Examples

The following SQL returns the exterior ring of a polygon as a closed linestring.

```
SELECT ST_AsEWKT(ST_ExteriorRing(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17 7,17 10,18 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12 14,15 14,13 11,12 14)))'));
```

```
st_asewkt

LINESTRING(7 9,8 7,11 6,15 8,16 6,17 7,17 10,18 12,17 14,15 15,11 15,10 13,9 12,7 9)
```

## ST\_Force2D

ST\_Force2D returns a 2D geometry of the input geometry. For 2D geometries, a copy of the input is returned. For 3DZ, 3DM, and 4D geometries, ST\_Force2D projects the geometry to the xy-Cartesian plane. Empty points in the input geometry remain empty points in the output geometry.

### Syntax

```
ST_Force2D(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

GEOMETRY.

The spatial reference system identifier (SRID) value of the returned geometry is the SRID value of the input geometry.

If *geom* is null, then null is returned.

If *geom* is empty, then an empty geometry is returned.

### Examples

The following SQL returns a 2D geometry from a 3DZ geometry.

```
SELECT ST_AsEWKT(ST_Force2D(ST_GeomFromText('MULTIPOINT Z(0 1 2, EMPTY, 2 3 4, 5 6 7)')));
```

```
st_asewkt

MULTIPOINT((0 1),EMPTY,(2 3),(5 6))
```

## ST\_Force3D

ST\_Force3D is an alias for ST\_Force3DZ. For more information, see [ST\\_Force3DZ \(p. 1218\)](#).

## ST\_Force3DM

ST\_Force3DM returns a 3DM geometry of the input geometry. For 2D geometries, the m coordinates of the nonempty points in the output geometry are all set to 0. For 3DM geometries, a copy of the input geometry is returned. For 3DZ geometries, the geometry is projected to the xy-Cartesian plane, and the m coordinates of the nonempty points in the output geometry are all set to 0. For 4D geometries, the geometry is projected to the xym-Cartesian space. Empty points in the input geometry remain empty points in the output geometry.

### Syntax

```
ST_Force3DM(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

GEOMETRY.

The spatial reference system identifier (SRID) value of the returned geometry is the SRID value of the input geometry.

If *geom* is null, then null is returned.

If *geom* is empty, then an empty geometry is returned.

### Examples

The following SQL returns a 3DM geometry from a 3DZ geometry.

```
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromText('MULTIPOINT Z(0 1 2, EMPTY, 2 3 4, 5 6 7)')));
```

```
st_asewkt

MULTIPOINT M ((0 1 0),EMPTY,(2 3 0),(5 6 0))
```

## ST\_Force3DZ

ST\_Force3DZ returns a 3DZ geometry from the input geometry. For 2D geometries, the z coordinates of the nonempty points in the output geometry are all set to 0. For 3DM geometries, the geometry is projected on the xy-Cartesian plane, and the z coordinates of the nonempty points in the output geometry are all set to 0. For 3DZ geometries, a copy of the input geometry is returned. For 4D geometries, the geometry is projected to the xyz-Cartesian space. Empty points in the input geometry remain empty points in the output geometry.

### Syntax

```
ST_Force3DZ(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

GEOMETRY.

The spatial reference system identifier (SRID) value of the returned geometry is the SRID value of the input geometry.

If *geom* is null, then null is returned.

If *geom* is empty, then an empty geometry is returned.

## Examples

The following SQL returns a 3DZ geometry from a 3DM geometry.

```
SELECT ST_AsEWKT(ST_Force3DZ(ST_GeomFromText('MULTIPOINT M(0 1 2, EMPTY, 2 3 4, 5 6 7)')));
```

```
st_asewkt

MULTIPOINT Z ((0 1 0),EMPTY,(2 3 0),(5 6 0))
```

## ST\_Force4D

ST\_Force4D returns a 4D geometry of the input geometry. For 2D geometries, the z and m coordinates of the nonempty points in the output geometry are all set to 0. For 3DM geometries, the z coordinates of the nonempty points in the output geometry are all set to 0. For 3DZ geometries, the m coordinates of the nonempty points in the output geometry are all set to 0. For 4D geometries, a copy of the input geometry is returned. Empty points in the input geometry remain empty points in the output geometry.

## Syntax

```
ST_Force4D(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

GEOMETRY.

The spatial reference system identifier (SRID) value of the returned geometry is the SRID value of the input geometry.

If *geom* is null, then null is returned.

If *geom* is empty, then an empty geometry is returned.

## Examples

The following SQL returns a 4D geometry from a 3DM geometry.

```
SELECT ST_AsEWKT(ST_Force4D(ST_GeomFromText('MULTIPOINT M(0 1 2, EMPTY, 2 3 4, 5 6 7)')));

st_asewkt

MULTIPOINT ZM ((0 1 0 2),EMPTY,(2 3 0 4),(5 6 0 7))
```

## ST\_GeoHash

*ST\_GeoHash* returns the geohash representation of the input point with the specified precision. The default precision value is 20. For more information about the definition of geohash, see [Geohash](#) in Wikipedia.

### Syntax

```
ST_GeoHash(geom)
```

```
ST_GeoHash(geom, precision)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*precision*

A value of data type INTEGER. The default is 20.

### Return type

GEOMETRY

The function returns the geohash representation of the input point.

If the input point is empty, the function returns null.

If the input geometry is not a point, the function returns an error.

## Examples

The following SQL returns the geohash representation of the input point.

```
SELECT ST_GeoHash(ST_GeomFromText('POINT(45 -45)'), 25) AS geohash;

geohash

m00000000000000000000000gzz
```

The following SQL returns null because the input point is empty.

```
SELECT ST_GeoHash(ST_GeomFromText('POINT EMPTY'), 10) IS NULL AS result;
```

```
result

true
```

## ST\_GeogFromText

`ST_GeogFromText` constructs a geography object from a well-known text (WKT) or extended well-known text (EWKT) representation of an input geography.

### Syntax

```
ST_GeogFromText(wkt_string)
```

### Arguments

*wkt\_string*

A value of data type VARCHAR that is a WKT or EWKT representation of a geography.

### Return type

GEOGRAPHY

If the SRID value is set to the provided value in the input. If SRID is not provided, it is set to 4326.

If *wkt\_string* is null, then null is returned.

If *wkt\_string* is not valid, then an error is returned.

### Examples

The following SQL constructs a polygon from a geography object with an SRID value.

```
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4324;POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))'));
```

```
st_asewkt

SRID=4324;POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))
```

The following SQL constructs a polygon from a geography object. The SRID value is set to 4326.

```
SELECT ST_AsEWKT(ST_GeogFromText('POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))'));
```

```
st_asewkt

SRID=4326;POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))
```

## ST\_GeogFromWKB

`ST_GeogFromWKB` constructs a geography object from a hexadecimal well-known binary (WKB) representation of an input geography.

## Syntax

`ST_GeogFromWKB(wkb_string)`

## Arguments

*wkb\_string*

A value of data type VARCHAR that is a hexadecimal WKB representation of a geography.

## Return type

## GEOGRAPHY

If the SRID value is provided it is set to the provided value. If SRID is not provided, it is set to 4326.

If `wkb_string` is null, then null is returned.

If `wkb_string` is not valid, then an error is returned.

## Examples

The following SQL constructs a geography from a hexadecimal WKB value.

st\_asewkt

---

ST GeometryN

`ST_GeometryN` returns a geometry pointed to by the input index of the input geometry, as follows:

- If the input is a point, linestring, or polygon, then a geometry is returned as is if the index is equal to one (1), and null if the index is other than one (1).
  - If the input is a multipoint, multilinestring, multipolygon, or geometry collection, then a point, linestring, polygon, or geometry collection is returned as pointed to by an input index.

The index is one-based. The spatial reference system identifier (SRID) of the result is the same as that of the input geometry. The dimension of the returned geometry is the same as that of the input geometry.

## Syntax

**ST\_GeometryN(*geom*, *index*)**

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*index*

A value of data type INTEGER that represents the position of a one-based index.

## Return type

GEOMETRY

If *geom* or *index* is null, then null is returned.

If *index* is out of range, then an error is returned.

## Examples

The following SQL returns the geometries in a geometry collection.

```
WITH tmp1(idx) AS (SELECT 1 UNION SELECT 2),
tmp2(g) AS (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0 10,0
0)),LINESTRING(20 10,20 0,10 0))'))
SELECT idx, ST_AsEWKT(ST_GeometryN(g, idx)) FROM tmp1, tmp2 ORDER BY idx;
```

| idx | st_asewkt                    |
|-----|------------------------------|
| 1   | POLYGON((0 0,10 0,0 10,0 0)) |
| 2   | LINESTRING(20 10,20 0,10 0)  |

## ST\_GeometryType

ST\_GeometryType returns the subtype of an input geometry as a string.

For 3DM, 3DZ, and 4D geometry inputs, ST\_GeometryType returns the same result as for 2D geometry inputs.

## Syntax

```
ST_GeometryType(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

VARCHAR representing the subtype of *geom*.

If *geom* is null, then null is returned.

The values returned are as follows.

| Returned string value | Geometry subtype                                        |
|-----------------------|---------------------------------------------------------|
| ST_Point              | Returned if <i>geom</i> is a POINT subtype              |
| ST_LineString         | Returned if <i>geom</i> is a LINESTRING subtype         |
| ST_Polygon            | Returned if <i>geom</i> is a POLYGON subtype            |
| ST_MultiPoint         | Returned if <i>geom</i> is a MULTIPOLYPOINT subtype     |
| ST_MultiLineString    | Returned if <i>geom</i> is a MULTILINESTRING subtype    |
| ST_MultiPolygon       | Returned if <i>geom</i> is a MULTIPOLYGON subtype       |
| ST_GeometryCollection | Returned if <i>geom</i> is a GEOMETRYCOLLECTION subtype |

## Examples

The following SQL returns the subtype of the input linestring geometry.

```
SELECT ST_GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
29.31,77.29 29.07)'));
```

```
st_geometrytype

ST_LineString
```

## ST\_GeomFromEWKB

`ST_GeomFromEWKB` constructs a geometry object from the extended well-known binary (EWKB) representation of an input geometry.

`ST_GeomFromEWKB` accepts 3DZ, 3DM, and 4D geometries written in WKB and EWKB hexadecimal format.

### Syntax

```
ST_GeomFromEWKB(ewkb_string)
```

### Arguments

*ewkb\_string*

A value of data type VARCHAR that is a hexadecimal EWKB representation of a geometry.

### Return type

GEOMETRY

If *ewkb\_string* is null, then null is returned.

If *ewkb\_string* is not valid, then an error is returned.

## Examples

The following SQL constructs a polygon from an EWKB value and returns the EWKT representation of a polygon.

st\_asewkt

## ST\_GeomFromEWKT

`ST_GeomFromEWKT` constructs a geometry object from the extended well-known text (EWKT) representation of an input geometry.

`ST_GeomFromEWKT` accepts 3DZ, 3DM, and 4D where the geometry type is prefixed with Z, M, or ZM, respectively.

## Syntax

`ST_GeomFromEWKT(ewkt_string)`

## Arguments

### *ewkt string*

A value of data type VARCHAR or an expression that evaluates to a VARCHAR type, that is an EWKT representation of a geometry.

You can use the WKT keyword EMPTY to designate an empty point, a multipoint with an empty point, or a geometry collection with an empty point. The following example creates an empty point.

```
ST_GeomFromEWKT('SRID=4326;POINT EMPTY');
```

## Return type

GEOMETRY

If `ewkt_string` is null, then null is returned.

If `ewkt` string is not valid, then an error is returned.

## Examples

The following SQL constructs a multilinestring from an EWKT value and returns a geometry. It also returns the ST\_AsEWKT result of the geometry.

```
SELECT ST_GeomFromEWKT('SRID=4326;MULTILINESTRING((1 0,1 0),(2 0,3 0),(4 0,5 0,6 0))') as geom, ST_AsEWKT(geom);
```

```
 geom
 |
 st_asewkt

+-----+
0105000020E61000000300000001020000000200000000000000000000F03F00000000000000000000000000000000000000F03F00000000000
| SRID=4326;MULTILINESTRING((1 0,1 0),(2 0,3 0),(4 0,5 0,6 0))
```

## ST\_GeomFromGeoJSON

ST\_GeomFromGeoJSON constructs a geometry object from the GeoJSON representation of an input geometry. For more information about the GeoJSON format, see [GeoJSON](#) in Wikipedia.

If there is at least one point with three or more coordinates, the resulting geometry is 3DZ, where the Z component is zero for the points that have only two coordinates. If all points in the input GeoJSON contain two coordinates or are empty, ST\_GeomFromGeoJSON returns a 2D geometry. The returned geometry always has the spatial reference identifier (SRID) of 4326.

### Syntax

```
ST_GeomFromGeoJSON(geojson_string)
```

### Arguments

*geojson\_string*

A value of data type VARCHAR or an expression that evaluates to a VARCHAR type, that is a GeoJSON representation of a geometry.

### Return type

GEOMETRY

If *geojson\_string* is null, then null is returned.

If *geojson\_string* is not valid, then an error is returned.

### Examples

The following SQL returns a 2D geometry represented in the input GeoJSON.

```
SELECT ST_AsEWKT(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[1,2]}'));
```

```
st_asewkt

SRID=4326;POINT(1 2)
```

The following SQL returns a 3DZ geometry represented in the input GeoJSON.

```
SELECT ST_AsEWKT(ST_GeomFromGeoJSON('{"type":"LineString","coordinates":[[1,2,3],[4,5,6],[7,8,9]]}'));
```

```
st_asewkt

SRID=4326;LINESTRING Z (1 2 3,4 5 6,7 8 9)
```

The following SQL returns 3DZ geometry when only one point has three coordinates while all other points have two coordinates in the input GeoJSON.

```
SELECT ST_AsEWKT(ST_GeomFromGeoJSON('{"type":"Polygon","coordinates": [[[0, 0],[0, 1, 8],[1, 0],[0, 0]]]}'));
```

```
st_asewkt

SRID=4326;POLYGON Z ((0 0 0,0 1 8,1 0 0,0 0 0))
```

## ST\_GeomFromGeoHash

`ST_GeomFromGeoHash` constructs a geometry object from the geohash representation of an input geometry. `ST_GeomFromGeoHash` returns a two-dimensional (2D) geometry with the spatial reference identifier (SRID) of zero (0). For more information about the geohash format, see [Geohash](#) in Wikipedia.

### Syntax

```
ST_GeomFromGeoHash(geohash_string)
```

```
ST_GeomFromGeoHash(geohash_string, precision)
```

### Arguments

*geohash\_string*

A value of data type VARCHAR or an expression that evaluates to a VARCHAR type, that is a geohash representation of a geometry.

*precision*

A value of data type INTEGER that represents the precision of the geohash. The value is the number of characters of the geohash to be used as precision. If the value is not specified, less than zero, or greater than the *geohash\_string* length, then the *geohash\_string* length is used.

### Return type

GEOMETRY

If *geohash\_string* is null, then null is returned.

If *geohash\_string* is not valid, then an error is returned.

### Examples

The following SQL returns a polygon with high precision.

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0'));
```

```
st_asewkt

POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646))
```

The following SQL returns a point with high precision.

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz00'));
```

```
st_asewkt

POINT(-115.172816 36.114646)
```

The following SQL returns a polygon with low precision.

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qq'));
```

```
st_asewkt

POLYGON((-115.3125 35.15625,-115.3125 36.5625,-113.90625 36.5625,-113.90625 35.15625,-115.3125 35.15625))
```

The following SQL returns a polygon with precision 3.

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 3));
```

```
st_asewkt

POLYGON((-115.3125 35.15625,-115.3125 36.5625,-113.90625 36.5625,-113.90625 35.15625,-115.3125 35.15625))
```

## ST\_GeomFromText

`ST_GeomFromText` constructs a geometry object from a well-known text (WKT) representation of an input geometry.

`ST_GeomFromText` accepts 3DZ, 3DM, and 4D where the geometry type is prefixed with Z, M, or ZM, respectively.

### Syntax

```
ST_GeomFromText(wkt_string)
```

```
ST_GeomFromText(wkt_string, srid)
```

## Arguments

*wkt\_string*

A value of data type VARCHAR that is a WKT representation of a geometry.

You can use the WKT keyword EMPTY to designate an empty point, a multipoint with an empty point, or a geometry collection with an empty point. The following example creates a multipoint with one empty and one nonempty point.

```
ST_GeomFromEWKT('MULTIPOINT(1 0,EMPTY)');
```

*srid*

A value of data type INTEGER that is a spatial reference identifier (SRID). If an SRID value is provided, the returned geometry has this SRID value. Otherwise, the SRID value of the returned geometry is set to zero (0).

## Return type

## GEOMETRY

If `wkt_string` or `srid` is null, then null is returned.

If *srid* is negative, then null is returned.

If `wkt_string` is not valid, then an error is returned.

If *srid* is not valid, then an error is returned.

## Examples

The following SQL constructs a geometry object from the WKT representation and SRID value.

```
SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326);
```

## st\_geomfromtext

0103000030E610000000100000000E0000

## ST\_GeomFromWKB

`ST_GeomFromWKB` constructs a geometry object from a hexadecimal well-known binary (WKB) representation of an input geometry.

`ST_GeomFromWKB` accepts 3DZ, 3DM, and 4D geometries written in WKB hexadecimal format.

## Syntax

`ST_GeomFromWKB(wkb_string)`

```
ST_GeomFromWKB(wkb_string, srid)
```

## Arguments

*wkb\_string*

A value of data type VARCHAR that is a hexadecimal WKB representation of a geometry.

*srid*

A value of data type INTEGER that is a spatial reference identifier (SRID). If an SRID value is provided, the returned geometry has this SRID value. Otherwise, the SRID value of the returned geometry is set to 0.

## Return type

GEOMETRY

If `wkb_string` or `srid` is null, then null is returned.

If *srid* is negative, then null is returned.

If `wkb_string` is not valid, then an error is returned.

If *srid* is not valid, then an error is returned.

## Examples

The following SQL constructs a polygon from a WKB value and returns the WKT representation of a polygon.

```
st_astext
```

## ST\_InteriorRingN

`ST_InteriorRingN` returns a closed linestring corresponding to the interior ring of an input polygon at the index position. The dimension of the returned geometry is the same as that of the input geometry.

## Syntax

`ST_InteriorRingN(geom, index)`

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### *index*

A value of data type INTEGER that represents the position of a ring of a one-based index.

### Return type

GEOMETRY of subtype LINESTRING.

The spatial reference system identifier (SRID) value of the returned geometry is the SRID value of the input geometry.

If *geom* or *index* is null, then null is returned.

If *index* is out of range, then null is returned.

If *geom* is not a polygon, then null is returned.

If *geom* is an empty polygon, then null is returned.

### Examples

The following SQL returns the second ring of the polygon as a closed linestring.

```
SELECT ST_AsEWKT(ST_InteriorRingN(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17 7,17 10,18 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12 14,15 14,13 11,12 14))'),2));
```

```
st_asewkt

LINESTRING(12 14,15 14,13 11,12 14)
```

## ST\_Intersects

ST\_Intersects returns true if the 2D projections of the two input geometries have at least one point in common.

### Syntax

```
ST_Intersects(geom1, geom2)
```

### Arguments

#### *geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

#### *geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

BOOLEAN

If *geom1* or *geom2* is null, then null is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geom1* or *geom2* is a geometry collection, then an error is returned.

## Examples

The following SQL checks if the first polygon intersects the second polygon.

```
SELECT ST_Intersects(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(2 2,2 5,5 5,5 2,2 2))'), ST_GeomFromText('MULTIPOINT((4 4),(6 6))'));
```

```
st_intersects

true
```

## ST\_Intersection

*ST\_Intersection* returns a geometry representing the point-set intersection of two geometries. That is, it returns the portion of the two input geometries that is shared between them.

### Syntax

```
ST_Intersection(geom1, geom2)
```

### Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

GEOMETRY

If *geom1* and *geom2* don't share any space (they are disjoint), then an empty geometry is returned.

If *geom1* or *geom2* are empty, then an empty geometry is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geom1* or *geom2* is a geometry collection, then an error is returned.

If *geom1* or *geom2* is not a two-dimensional (2D) geometry, then an error is returned.

## Examples

The following SQL returns the non-empty geometry representing the intersection of two input geometries.

```
SELECT ST_AsEWKT(ST_Intersection(ST_GeomFromText('polygon((0 0,100 100,0 200,0 0))'),
ST_GeomFromText('polygon((0 0,10 0,0 10,0 0))')));
```

```
st_asewkt

POLYGON((0 0,0 10,5 5,0 0))
```

The following SQL returns an empty geometry when passed disjoint (non-intersecting) input geometries.

```
SELECT ST_AsEWKT(ST_Intersection(ST_GeomFromText('linestring(0 100,0 0)'),
ST_GeomFromText('polygon((1 0,10 0,1 10,1 0))')));
```

```
st_asewkt

LINESTRING EMPTY
```

## ST\_IsPolygonCCW

`ST_IsPolygonCCW` returns true if the 2D projection of the input polygon or multipolygon is counterclockwise. If the input geometry is a point, linestring, multipoint, or multilinestring, then true is returned. For geometry collections, `ST_IsPolygonCCW` returns true if all the geometries in the collection are counterclockwise.

### Syntax

```
ST_IsPolygonCCW(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

BOOLEAN

If *geom* is null, then null is returned.

### Examples

The following SQL checks if the polygon is counterclockwise.

```
SELECT ST_IsPolygonCCW(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17 7,17 10,18 12,17
14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12 14,15 14,13 11,12
14))'));
```

```
st_ispolygonccw

true
```

## ST\_IsPolygonCW

`ST_IsPolygonCW` returns true if the 2D projection of the input polygon or multipolygon is clockwise. If the input geometry is a point, linestring, multipoint, or multilinestring, then true is returned. For geometry collections, `ST_IsPolygonCW` returns true if all the geometries in the collection are clockwise.

### Syntax

```
ST_IsPolygonCW(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

BOOLEAN

If *geom* is null, then null is returned.

### Examples

The following SQL checks if the polygon is clockwise.

```
SELECT ST_IsPolygonCW(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17 7,17 10,18 12,17
14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12 14,15 14,13 11,12
14))'));
```

```
st_ispolygonccw

true
```

## ST\_IsClosed

`ST_IsClosed` returns true if the 2D projection of the input geometry is closed. The following rules define a closed geometry:

- The input geometry is a point or a multipoint.
- The input geometry is a linestring, and the start and end points of the linestring coincide.
- The input geometry is a nonempty multilinestring and all its linestrings are closed.
- The input geometry is a nonempty polygon, all polygon's rings are nonempty, and the start and end points of all its rings coincide.
- The input geometry is a nonempty multipolygon and all its polygons are closed.
- The input geometry is a nonempty geometry collection and all its components are closed.

### Syntax

```
ST_IsClosed(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

BOOLEAN

If *geom* is an empty point, then false is returned.

If *geom* is null, then null is returned.

## Examples

The following SQL checks if the polygon is closed.

```
SELECT ST_IsClosed(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
st_isclosed

true
```

## ST\_IsCollection

ST\_IsCollection returns true if the input geometry is one of the following subtypes: GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, or MULTIPOLYGON.

## Syntax

```
ST_IsCollection(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

BOOLEAN

If *geom* is null, then null is returned.

## Examples

The following SQL checks if the polygon is a collection.

```
SELECT ST_IsCollection(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
st_iscollection

```

```
false
```

## ST\_IsEmpty

ST\_IsEmpty returns true if the input geometry is empty. A geometry is not empty if it contains at least one nonempty point.

ST\_IsEmpty returns true if the input geometry has at least one nonempty point.

### Syntax

```
ST_IsEmpty(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

BOOLEAN

If *geom* is null, then null is returned.

### Examples

The following SQL checks if the specified polygon is empty.

```
SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
st_isempty

false
```

## ST\_IsRing

ST\_IsRing returns true if the input linestring is a ring. A linestring is a ring if it is closed and simple.

### Syntax

```
ST_IsRing(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The geometry must be a LINESTRING.

### Return type

BOOLEAN

If *geom* is not a LINESTRING, then an error is returned.

## Examples

The following SQL checks if the specified linestring is a ring.

```
SELECT ST_IsRing(ST_GeomFromText('linestring(0 0, 1 1, 1 2, 0 0)'));
```

```
st_isring

true
```

## ST\_IsSimple

ST\_IsSimple returns true if the 2D projection of the input geometry is simple. For more information about the definition of a simple geometry, see [Geometric simplicity \(p. 231\)](#).

### Syntax

```
ST_IsSimple(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

BOOLEAN

If *geom* is null, then null is returned.

## Examples

The following SQL checks if the specified linestring is simple. In this example, it isn't simple because it has self-intersection.

```
SELECT ST_IsSimple(ST_GeomFromText('LINESTRING(0 0,10 0,5 5,5 -5)'));
```

```
st_issimple

false
```

## ST\_IsValid

ST\_IsValid returns true if the 2D projection of the input geometry is valid. For more information about the definition of a valid geometry, see [Geometric validity \(p. 229\)](#).

### Syntax

```
ST_IsValid(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

BOOLEAN

If *geom* is null, then null is returned.

## Examples

The following SQL checks if the specified polygon is valid. In this example, the polygon is invalid because the interior of the polygon isn't simply connected.

```
SELECT ST_IsValid(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(5 0,10 5,5 10,0 5,5 0))'));
```

```
st_isvalid

false
```

## ST\_Length

For a linear geometry, ST\_Length returns the Cartesian length of a 2D projection. The length units are the same as the units in which the coordinates of the input geometry are expressed. The function returns zero (0) for points, multipoints, and areal geometries. When the input is a geometry collection, the function returns the sum of the lengths of the geometries in the collection.

For a geography, ST\_Length returns the geodesic length of the 2D projection of an input linear geography computed on the spheroid determined by the SRID. The unit of length is in meters. The function returns zero (0) for points, multipoints, and areal geographies. When the input is a geometry collection, the function returns the sum of the lengths of the geographies in the collection.

## Syntax

```
ST_Length(geo)
```

## Arguments

*geo*

A value of data type GEOMETRY or GEOGRAPHY, or an expression that evaluates to a GEOMETRY or GEOGRAPHY type.

## Return type

DOUBLE PRECISION

If *geo* is null, then null is returned.

If the SRID value is not found, then an error is returned.

## Examples

The following SQL returns the Cartesian length of a multilinestring.

```
SELECT ST_Length(ST_GeomFromText('MULTILINESTRING((0 0,10 0,0 10),(10 0,20 0,20 10))'));
```

```
st_length

44.142135623731
```

The following SQL returns the length of a linestring in a geography.

```
SELECT ST_Length(ST_GeogFromText('SRID=4326;LINESTRING(5 0,6 0,4 0)'));
```

```
st_length

333958.472379804
```

The following SQL returns the length of a point in a geography.

```
SELECT ST_Length(ST_GeogFromText('SRID=4326;POINT(4 5)'));
```

```
st_length

0
```

## ST\_LengthSphere

ST\_LengthSphere returns the length of a linear geometry in meters. For point, multipoint, and areal geometries, ST\_LengthSphere returns 0. For geometry collections, ST\_LengthSphere returns the total length of the linear geometries in the collection in meters.

ST\_LengthSphere interprets the coordinates of each point of the input geometry as longitude and latitude in degrees. For 3DZ, 3DM, or 4D geometries, only the first two coordinates are used.

### Syntax

```
ST_LengthSphere(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

DOUBLE PRECISION length in meters. The length computation is based on the spherical model of the Earth whose radius is Earth's mean radius of the World Geodetic System (WGS) 84 ellipsoidal model of the Earth.

If *geom* is null, then null is returned.

## Examples

The following example SQL computes the length of a linestring in meters.

```
SELECT ST_LengthSphere(ST_GeomFromText('LINESTRING(10 10,45 45)'));
```

```
st_lengthsphere

5127736.0829256
```

## ST\_Length2D

ST\_Length2D is an alias for ST\_Length. For more information, see [ST\\_Length \(p. 1238\)](#).

## ST\_LineFromMultiPoint

ST\_LineFromMultiPoint returns a linestring from an input multipoint geometry. The order of the points is preserved. The spatial reference system identifier (SRID) of the returned geometry is the same as that of the input geometry. The dimension of the returned geometry is the same as that of the input geometry.

### Syntax

```
ST_LineFromMultiPoint(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be MULTIPOLYPOINT.

### Return type

GEOMETRY

If *geom* is null, then null is returned.

If *geom* is empty, then an empty LINESTRING is returned.

If *geom* contains empty points, then these empty points are ignored.

If *geom* isn't a MULTIPOLYPOINT, then error is returned.

## Examples

The following SQL creates a linestring from a multipoint.

```
SELECT ST_AsEWKT(ST_LineFromMultiPoint(ST_GeomFromText('MULTIPOINT(0 0,10 0,10 10,5 5,0 5)',4326)));
```

```
st_asewkt

SRID=4326;LINESTRING(0 0,10 0,10 10,5 5,0 5)
```

## ST\_LineInterpolatePoint

`ST_LineInterpolatePoint` returns a point along a line at a fractional distance from the start of the line.

To determine point equality, `ST_LineInterpolatePoint` operates on the 2D projection of the input geometry. If the input geometry is empty, a copy of it is returned in the same dimension as the input. For 3DZ, 3DM, and 4D geometries, the z or m coordinate is the average of the z or m coordinates of the segment where the point lies.

### Syntax

```
ST_LineInterpolatePoint(geom, fraction)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype is LINESTRING.

*fraction*

A value of data type DOUBLE PRECISION that represents the position of a point along the linestring for the line. The value is a fraction in the range 0–1, inclusive.

### Return type

GEOMETRY of subtype POINT.

If *geom* or *fraction* is null, then null is returned.

If *geom* is empty, then the empty point is returned.

The spatial reference system identifier (SRID) value of the returned geometry is the SRID value of the input geometry.

If *fraction* is out of range, then an error is returned.

If *geom* is not a linestring, then an error is returned.

### Examples

The following SQL returns a point halfway along a linestring.

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(ST_GeomFromText('LINESTRING(0 0, 5 5, 7 7, 10 10)'), 0.50));
```

```
st_asewkt

POINT(5 5)
```

The following SQL returns a point 90 percent of the way along a linestring.

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(ST_GeomFromText('LINESTRING(0 0, 5 5, 7 7, 10 10)'), 0.90));
```

```
st_asewkt

POINT(9 9)
```

## ST\_M

ST\_M returns the m coordinate of an input point.

### Syntax

```
ST_M(point)
```

### Arguments

*point*

A POINT value of data type GEOMETRY.

### Return type

DOUBLE PRECISION value of the m coordinate.

If *point* is null, then null is returned.

If *point* is a 2D or 3DZ point, then null is returned.

If *point* is the empty point, then null is returned.

If *point* is not a POINT, then an error is returned.

### Examples

The following SQL returns the m coordinate of a point in a 3DM geometry.

```
SELECT ST_M(ST_GeomFromEWKT('POINT M (1 2 3)'));
```

```
st_m

3
```

The following SQL returns the m coordinate of a point in a 4D geometry.

```
SELECT ST_M(ST_GeomFromEWKT('POINT ZM (1 2 3 4)'));
```

```
st_m
```

4

## ST\_MakeEnvelope

ST\_MakeEnvelope returns a geometry as follows:

- If the input coordinates specify a point, then the returned geometry is a point.
- If the input coordinates specify a line, then the returned geometry is a linestring.
- Otherwise, the returned geometry is a polygon, where the input coordinates specify the lower-left and upper-right corners of a box.

If provided, the spatial reference system identifier (SRID) value of the returned geometry is set to the input SRID value.

### Syntax

```
ST_MakeEnvelope(xmin, ymin, xmax, ymax)
```

```
ST_MakeEnvelope(xmin, ymin, xmax, ymax, srid)
```

### Arguments

*xmin*

A value of data type DOUBLE PRECISION. This value is the first coordinate of the lower-left corner of a box.

*ymin*

A value of data type DOUBLE PRECISION. This value is the second coordinate of the lower-left corner of a box.

*xmax*

A value of data type DOUBLE PRECISION. This value is the first coordinate of the upper-right corner of a box.

*ymax*

A value of data type DOUBLE PRECISION. This value is the second coordinate of the upper-right corner of a box.

*srid*

A value of data type INTEGER that represents a spatial reference system identifier (SRID). If the SRID value is not provided, then it is set to zero.

### Return type

GEOMETRY of subtype POINT, LINESTRING, or POLYGON.

The SRID of the returned geometry is set to *srid* or zero if *srid* isn't set.

If *xmin*, *ymin*, *xmax*, *ymax*, or *srid* is null, then null is returned.

If *srid* is negative, then an error is returned.

## Examples

The following SQL returns a polygon representing an envelope defined by the four input coordinate values.

```
SELECT ST_AsEWKT(ST_MakeEnvelope(2,4,5,7));
```

```
st_astext

POLYGON((2 4,2 7,5 7,5 4,2 4))
```

The following SQL returns a polygon representing an envelope defined by the four input coordinate values and an SRID value.

```
SELECT ST_AsEWKT(ST_MakeEnvelope(2,4,5,7,4326));
```

```
st_astext

SRID=4326;POLYGON((2 4,2 7,5 7,5 4,2 4))
```

## ST\_MakeLine

ST\_MakeLine creates a linestring from the input geometries.

The dimension of the returned geometry is the same as that of the input geometries. Both input geometries must of the same dimension.

### Syntax

```
ST_MakeLine(geom1, geom2)
```

### Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be POINT, LINESTRING, or MULTIPOLYPOINT.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be POINT, LINESTRING, or MULTIPOLYPOINT.

### Return type

GEOMETRY of subtype LINESTRING.

If *geom1* or *geom2* is null, then null is returned.

If *geom1* and *geom2* is the empty point or contains empty points, then these empty points are ignored.

If *geom1* and *geom2* are empty, then the empty LINESTRING is returned.

The spatial reference system identifier (SRID) value of the returned geometry is the SRID value of the input geometries.

If *geom1* and *geom2* have different SRID values, then an error is returned.

If *geom1* or *geom2* is not a POINT, LINESTRING, or MULTIPOLYPOINT, then an error is returned.

If *geom1* and *geom2* have different dimensions, then an error is returned.

## Examples

The following SQL constructs a linestring from two input linestrings.

```
SELECT ST_MakeLine(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'), ST_GeomFromText('LINESTRING(88.29 39.07,88.42 39.26,88.27 39.31,88.29 39.07)'));
```

```
st_makeline
```

```

```

```
010200000008000000C3F5285C8F52534052B81E85EB113D407B14AE47E15A5340C3F5285C8F423D40E17A14AE475153408FC2
```

## ST\_MakePoint

ST\_MakePoint returns a point geometry whose coordinate values are the input values.

### Syntax

```
ST_MakePoint(x, y)
```

```
ST_MakePoint(x, y, z)
```

```
ST_MakePoint(x, y, z, m)
```

### Arguments

x

A value of data type DOUBLE PRECISION representing the first coordinate.

y

A value of data type DOUBLE PRECISION representing the second coordinate.

z

A value of data type DOUBLE PRECISION representing the third coordinate.

m

A value of data type DOUBLE PRECISION representing the fourth coordinate.

### Return type

GEOMETRY of subtype POINT.

The spatial reference system identifier (SRID) value of the returned geometry is set to 0.

If  $x$ ,  $y$ ,  $z$ , or  $m$  is null, then null is returned.

## Examples

The following SQL returns a GEOMETRY type of subtype POINT with the provided coordinates.

```
SELECT ST_AsText(ST_MakePoint(1,3));
```

```
st_astext

POINT(1 3)
```

The following SQL returns a GEOMETRY type of subtype POINT with the provided coordinates.

```
SELECT ST_AsEWKT(ST_MakePoint(1, 2, 3));
```

```
st_asewkt

POINT Z (1 2 3)
```

The following SQL returns a GEOMETRY type of subtype POINT with the provided coordinates.

```
SELECT ST_AsEWKT(ST_MakePoint(1, 2, 3, 4));
```

```
st_asewkt

POINT ZM (1 2 3 4)
```

## ST\_MakePolygon

ST\_MakePolygon has two variants that return a polygon. One takes a single geometry, and another takes two geometries.

- The input of the first variant is a linestring that defines the outer ring of the output polygon.
- The input of the second variant is a linestring and a multilinestring. Both of these are empty or closed.

The boundary of the exterior ring of the output polygon is the input linestring, and the boundaries of the interior rings of the polygon are the linestrings in the input multilinestring. If the input linestring is empty, an empty polygon is returned. Empty linestrings in the multilinestring are disregarded. The spatial reference system identifier (SRID) of the resulting geometry is the common SRID of the two input geometries.

The dimension of the returned geometry is the same as that of the input geometries. The exterior ring and interior rings must of the same dimension.

## Syntax

```
ST_MakePolygon(geom1)
```

```
ST_MakePolygon(geom1, geom2)
```

## Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be LINESTRING. The *linestring* value must be closed or empty.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be MULTILINESTRING.

## Return type

GEOMETRY of subtype POLYGON.

The spatial reference system identifier (SRID) of the returned geometry is equal to the SRID of the inputs.

If *geom1*, or *geom2* is null, then null is returned.

If *geom1* is not a linestring, then an error is returned.

If *geom2* is not a multilinestring, then an error is returned.

If *geom1* is not closed, then an error is returned.

If *geom1* is a single point or is not closed, then an error is returned.

If *geom2* contains at least one linestring that has a single point or is not closed, then an error is returned.

If *geom1* and *geom2* have different SRID values, then an error is returned.

If *geom1* and *geom2* have different dimensions, then an error is returned.

## Examples

The following SQL returns a polygon from an input linestring.

```
SELECT ST_AsText(ST_MakePolygon(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
```

```
st_astext

POLYGON((77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07))
```

The following SQL creates a polygon from a closed linestring and a closed multilinestring. The linestring is used for the exterior ring of the polygon. The linestrings in the multilinestrings are used for the interior rings of the polygon.

```
SELECT ST_AsEWKT(ST_MakePolygon(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,0 10,0 0)'),
ST_GeomFromText('MULTILINESTRING((1 1,1 2,2 1,1 1),(3 3,3 4,4 3,3 3)))');
```

```
st_astext

POLYGON((0 0,10 0,10 10,0 10,0 0),(1 1,1 2,2 1,1 1),(3 3,3 4,4 3,3 3))
```

## ST\_MemSize

ST\_MemSize returns the amount of memory space (in bytes) used by the input geometry. This size depends on the Amazon Redshift internal representation of the geometry and thus can change if the internal representation changes. You can use this size as an indication of the relative size of geometry objects in Amazon Redshift.

### Syntax

```
ST_MemSize(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

INTEGER representing the inherent dimension of *geom*.

If *geom* is null, then null is returned.

### Examples

The following SQL returns the memory size of a geometry collection.

```
SELECT ST_MemSize(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0 10,0 0)),LINESTRING(20 10,20 0,10 0))'))::varchar + ' bytes';
```

```
?column?

172 bytes
```

## ST\_MMax

ST\_MMax returns the maximum m coordinate of an input geometry.

### Syntax

```
ST_MMax(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

DOUBLE PRECISION value of the maximum m coordinate.

If *geom* is empty, then null is returned.

If *geom* is null, then null is returned.

If *geom* is a 2D or 3DZ geometry, then null is returned.

## Examples

The following SQL returns the largest m coordinate of a linestring in a 3DM geometry.

```
SELECT ST_MMax(ST_GeomFromEWKT('LINESTRING M (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_mmax
```

```

```

```
8
```

The following SQL returns the largest m coordinate of a linestring in a 4D geometry.

```
SELECT ST_MMax(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_mmax
```

```

```

```
11
```

## ST\_MMin

*ST\_MMin* returns the minimum m coordinate of an input geometry.

### Syntax

```
ST_MMin(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

DOUBLE PRECISION value of the minimum m coordinate.

If *geom* is empty, then null is returned.

If *geom* is null, then null is returned.

If *geom* is a 2D or 3DZ geometry, then null is returned.

## Examples

The following SQL returns the smallest m coordinate of a linestring in a 3DM geometry.

```
SELECT ST_MMin(ST_GeomFromEWKT('LINESTRING M (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_mmin

2
```

The following SQL returns the smallest m coordinate of a linestring in a 4D geometry.

```
SELECT ST_MMin(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_mmin

3
```

## ST\_Multi

`ST_Multi` converts a geometry to the corresponding multitype. If the input geometry is already a multitype or a geometry collection, a copy of it is returned. If the input geometry is a point, linestring, or polygon, then a multipoint, multilinestring, or multipolygon, respectively, that contains the input geometry is returned.

### Syntax

```
ST_Multi(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

GEOMETRY with subtype MULTIPOLYPOINT, MULTILINESTRING, MULTIPOLYGON, or GEOMETRYCOLLECTION.

The spatial reference system identifier (SRID) of the returned geometry is the same as that of the input geometry.

If *geom* is null, then null is returned.

### Examples

The following SQL returns a multipoint from an input multipoint.

```
SELECT ST_AsEWKT(ST_Multi(ST_GeomFromText('MULTIPOINT((1 2),(3 4))', 4326)));
```

```
st_asewkt

SRID=4326;MULTIPOINT((1 2),(3 4))
```

The following SQL returns a multipoint from an input point.

```
SELECT ST_AsEWKT(ST_Multi(ST_GeomFromText('POINT(1 2)', 4326)));
```

```
st_asewkt

SRID=4326;MULTIPOINT((1 2))
```

The following SQL returns a geometry collection from an input geometry collection.

```
SELECT ST_AsEWKT(ST_Multi(ST_GeomFromText('GEOMETRYCOLLECTION(POINT(1 2),MULTIPOINT((1 2),(3 4)))', 4326)));
```

```
st_asewkt

SRID=4326;GEOMETRYCOLLECTION(POINT(1 2),MULTIPOINT((1 2),(3 4)))
```

## ST\_NDims

`ST_NDims` returns the coordinate dimension of a geometry. `ST_NDims` doesn't consider the topological dimension of a geometry. Instead, it returns a constant value depending on the dimension of the geometry.

### Syntax

```
ST_NDims(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

INTEGER representing the inherent dimension of *geom*.

If *geom* is null, then null is returned.

The values returned are as follows.

| Returned value | Dimension of input geometry |
|----------------|-----------------------------|
| 2              | 2D                          |
| 3              | 3DZ or 3DM                  |
| 4              | 4D                          |

### Examples

The following SQL returns the number of dimensions of a 2D linestring.

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING(0 0,1 1,2 2,0 0)'));
```

```
st_ndims

2
```

The following SQL returns the number of dimensions of a 3DZ linestring.

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING Z(0 0 3,1 1 3,2 2 3,0 0 3)'));
```

```
st_ndims

3
```

The following SQL returns the number of dimensions of a 3DM linestring.

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING M(0 0 4,1 1 4,2 2 4,0 0 4)'));
```

```
st_ndims

3
```

The following SQL returns the number of dimensions of a 4D linestring.

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING ZM(0 0 3 4,1 1 3 4,2 2 3 4,0 0 3 4)'));
```

```
st_ndims

4
```

## ST\_NPoints

ST\_NPoints returns the number of nonempty points in an input geometry or geography.

### Syntax

```
ST_NPoints(geo)
```

### Arguments

*geo*

A value of data type GEOMETRY or GEOGRAPHY, or an expression that evaluates to a GEOMETRY or GEOGRAPHY type.

### Return type

INTEGER

If *geo* is an empty point, then 0 is returned.

If *geo* is null, then null is returned.

## Examples

The following SQL returns the number of points in a linestring.

```
SELECT ST_NPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
```

```
st_npoints

4
```

The following SQL returns the number of points in a linestring in a geography.

```
SELECT ST_NPoints(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
st_npoints

4
```

## ST\_NRings

*ST\_NRings* returns the number of rings in an input geometry.

### Syntax

```
ST_NRings(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

INTEGER

If *geom* is null, then null is returned.

The values returned are as follows.

| Returned value       | Geometry subtype                                                                           |
|----------------------|--------------------------------------------------------------------------------------------|
| 0                    | Returned if <i>geom</i> is a POINT, LINESTRING, MULTIPOLYPOINT, or MULTILINESTRING subtype |
| The number of rings. | Returned if <i>geom</i> is a POLYGON or MULTIPOLYGON subtype                               |

| Returned value                        | Geometry subtype                                        |
|---------------------------------------|---------------------------------------------------------|
| The number of rings in all components | Returned if <i>geom</i> is a GEOMETRYCOLLECTION subtype |

## Examples

The following SQL returns the number of rings in a multipolygon.

```
SELECT ST_NRings(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((0 0,-10 0,0 -10,0 0)))'));
```

```
st_nrings

2
```

## ST\_NumGeometries

ST\_NumGeometries returns the number of geometries in an input geometry.

### Syntax

```
ST_NumGeometries(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

INTEGER representing the number of geometries in *geom*.

If *geom* is null, then null is returned.

If *geom* is a single empty geometry, then 0 is returned.

If *geom* is a single nonempty geometry, then 1 is returned.

If *geom* is a GEOMETRYCOLLECTION or a MULTI subtype, then the number of geometries is returned.

## Examples

The following SQL returns the number of geometries in the input multilinestring.

```
SELECT ST_NumGeometries(ST_GeomFromText('MULTILINESTRING((0 0,1 0,0 5),(3 4,13 26)))'));
```

```
st_numgeometries

2
```

## ST\_NumInteriorRings

ST\_NumInteriorRings returns the number of rings in an input polygon geometry.

### Syntax

```
ST_NumInteriorRings(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

INTEGER

If *geom* is null, then null is returned.

If *geom* is not a polygon, then null is returned.

### Examples

The following SQL returns the number of interior rings in the input polygon.

```
SELECT ST_NumInteriorRings(ST_GeomFromText('POLYGON((0 0,100 0,100 100,0 100,0 0),(1 1,1 5,5 1,1 1),(7 7,7 8,8 7,7 7))'));
```

```
st_numinteriorrings

2
```

## ST\_NumPoints

ST\_NumPoints returns the number of points in an input geometry.

### Syntax

```
ST_NumPoints(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

INTEGER

If *geom* is null, then null is returned.

If *geom* is not of subtype LINESTRING, then null is returned.

## Examples

The following SQL returns the number of points in the input linestring.

```
SELECT ST_NumPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
```

```
st_numpoints

4
```

The following SQL returns null because the input *geom* is not of subtype LINESTRING.

```
SELECT ST_NumPoints(ST_GeomFromText('MULTIPOINT(1 2,3 4)'));
```

```
st_numpoints

```

## ST\_Perimeter

For an input areal geometry, ST\_Perimeter returns the Cartesian perimeter (length of the boundary) of the 2D projection. The perimeter units are the same as the units in which the coordinates of the input geometry are expressed. The function returns zero (0) for points, multipoints, and linear geometries. When the input is a geometry collection, the function returns the sum of the perimeters of the geometries in the collection.

For an input geography, ST\_Perimeter returns the geodesic perimeter (length of the boundary) of the 2D projection of an input areal geography computed on the spheroid determined by the SRID. The unit of perimeter is in meters. The function returns zero (0) for points, multipoints, and linear geographies. When the input is a geometry collection, the function returns the sum of the perimeters of the geographies in the collection.

## Syntax

```
ST_Perimeter(geo)
```

## Arguments

*geo*

A value of data type GEOMETRY or GEOGRAPHY, or an expression that evaluates to a GEOMETRY or GEOGRAPHY type.

## Return type

DOUBLE PRECISION

If *geo* is null, then null is returned.

If the SRID value is not found, then an error is returned.

## Examples

The following SQL returns the Cartesian perimeter of a multipolygon.

```
SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0),((10 0,20 0,20 10,10 0))))');
```

```
st_perimeter

68.2842712474619
```

The following SQL returns the Cartesian perimeter of a multipolygon.

```
SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0),((10 0,20 0,20 10,10 0))))');
```

```
st_perimeter

68.2842712474619
```

The following SQL returns the perimeter of a polygon in a geography.

```
SELECT ST_Perimeter(ST_GeogFromText('SRID=4326;POLYGON((0 0,1 0,0 1,0 0))'));
```

```
st_perimeter

378790.428393693
```

The following SQL returns the perimeter of a linestring in a geography.

```
SELECT ST_Perimeter(ST_GeogFromText('SRID=4326;LINESTRING(5 0,10 0)'));
```

```
st_perimeter

0
```

## ST\_Perimeter2D

ST\_Perimeter2D is an alias for ST\_Perimeter. For more information, see [ST\\_Perimeter \(p. 1256\)](#).

## ST\_Point

ST\_Point returns a point geometry from the input coordinate values.

### Syntax

```
ST_Point(x, y)
```

## Arguments

*x*

A value of data type DOUBLE PRECISION that represents a first coordinate.

*y*

A value of data type DOUBLE PRECISION that represents a second coordinate.

## Return type

GEOMETRY of subtype POINT.

The spatial reference system identifier (SRID) value of the returned geometry is set to 0.

If *x* or *y* is null, then null is returned.

## Examples

The following SQL constructs a point geometry from the input coordinates.

```
SELECT ST_AsText(ST_Point(5.0, 7.0));
```

```
st_astext

POINT(5 7)
```

## ST\_PointN

ST\_PointN returns a point in a linestring as specified by an index value. Negative index values are counted backward from the end of the linestring, so that -1 is the last point.

The dimension of the returned geometry is the same as that of the input geometry.

## Syntax

```
ST_PointN(geom, index)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be LINESTRING.

*index*

A value of data type INTEGER that represents the index of a point in a linestring.

## Return type

GEOMETRY of subtype POINT.

The spatial reference system identifier (SRID) value of the returned geometry is set to 0.

- If *geom* or *index* is null, then null is returned.
- If *index* is out of range, then null is returned.
- If *geom* is empty, then null is returned.
- If *geom* is not a LINestring, then null is returned.

## Examples

The following SQL returns an extended well-known text (EWKT) representation of a six-point LINestring to a GEOMETRY object and returns the point at index 5 of the linestring.

```
SELECT ST_AsEWKT(ST_PointN(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5,0 0)',4326),5));
```

```
st_asewkt

SRID=4326;POINT(0 5)
```

## ST\_Points

*ST\_Points* returns a multipoint geometry containing all nonempty points in the input geometry. *ST\_Points* doesn't remove points that are duplicated in the input, including the start and end points of ring geometries.

### Syntax

```
ST_Points(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

GEOMETRY of subtype MULTIPOLYLINE.

The spatial reference system identifier (SRID) value of the returned geometry is the same as *geom*.

If *geom* is null, then null is returned.

If *geom* is empty, then the empty multipoint is returned.

## Examples

The following SQL examples construct a multipoint geometry from the input geometry. The result is a multipoint geometry containing the nonempty points in the input geometry.

```
SELECT ST_AsEWKT(ST_Points(ST_SetSRID(ST_GeomFromText('LINESTRING(1 0,2 0,3 0)'), 4326)));
```

```
st_asewkt
```

```

SRID=4326;MULTIPOINT((1 0),(2 0),(3 0))
```

```
SELECT ST_AsEWKT(ST_Points(ST_SetSRID(ST_GeomFromText('MULTIPOLYGON(((0 0,1 0,0 1,0 0)))'),
4326)));
```

```
st_asewkt

SRID=4326;MULTIPOINT((0 0),(1 0),(0 1),(0 0))
```

## ST\_Polygon

`ST_Polygon` returns a polygon geometry whose outer ring is the input linestring with the value that was input for the spatial reference system identifier (SRID).

The dimension of the returned geometry is the same as that of the input geometry.

### Syntax

```
ST_Polygon(linestring, srid)
```

### Arguments

#### *linestring*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be LINESTRING that represents a linestring. The *linestring* value must be closed.

#### *srid*

A value of data type INTEGER that represents a SRID.

### Return type

GEOMETRY of subtype POLYGON.

The SRID value of the returned geometry is set to *srid*.

If *linestring* or *srid* is null, then null is returned.

If *linestring* is not a linestring, then an error is returned.

If *linestring* is not closed, then an error is returned.

If *srid* is negative, then an error is returned.

### Examples

The following SQL constructs a polygon with an SRID value.

```
SELECT ST_AsEWKT(ST_Polygon(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
29.31,77.29 29.07)'),4356));
```

```
st_asewkt
```

```

SRID=4356;POLYGON((77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07))
```

## ST\_RemovePoint

`ST_RemovePoint` returns a linestring geometry that has the point of the input geometry at an index position removed.

The index is zero-based. The spatial reference system identifier (SRID) of the result is the same as the input geometry. The dimension of the returned geometry is the same as that of the input geometry.

### Syntax

```
ST_RemovePoint(geom, index)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be LINESTRING.

*index*

A value of data type INTEGER that represents the position of a zero-based index.

### Return type

GEOMETRY

If *geom* or *index* is null, then null is returned.

If *geom* is not subtype LINESTRING, then an error is returned.

If *index* is out of range, then an error is returned. Valid values for the index position are between 0 and `ST_NumPoints(geom)` minus 1.

### Examples

The following SQL removes the last point in a linestring.

```
WITH tmp(g) AS (SELECT ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326))
SELECT ST_AsEWKT(ST_RemovePoint(g, ST_NumPoints(g) - 1)) FROM tmp;
```

```
st_asewkt
```

```

SRID=4326;LINESTRING(0 0,10 0,10 10,5 5)
```

## ST\_Reverse

`ST_Reverse` reverses the order of the vertices for linear and areal geometries. For point or multipoint geometries, a copy of the original geometry is returned. For geometry collections, `ST_Reverse` reverses the order of the vertices for each of the geometries in the collection.

The dimension of the returned geometry is the same as that of the input geometry.

## Syntax

```
ST_Reverse(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

GEOMETRY

The spatial reference system identifier (SRID) of the returned geometry is the same as that of the input geometry.

If *geom* is null, then null is returned.

## Examples

The following SQL reverses the order of the points in a linestring.

```
SELECT ST_AsEWKT(ST_Reverse(ST_GeomFromText('LINESTRING(1 0,2 0,3 0,4 0)', 4326)));
```

```
st_asewkt

SRID=4326;LINESTRING(4 0,3 0,2 0,1 0)
```

## ST\_SetPoint

ST\_SetPoint returns a linestring with updated coordinates with respect to the input linestring's position as specified by the index. The new coordinates are the coordinates of the input point.

The dimension of the returned geometry is the same as that of the *geom1* value. If *geom1* and *geom2* have different dimensions, *geom2* is projected to the dimension of *geom1*.

## Syntax

```
ST_SetPoint(geom1, index, geom2)
```

## Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be LINESTRING.

*index*

A value of data type INTEGER that represents the position of an index. A 0 refers to the first point of the linestring from the left, 1 refers to the second point, and so on. The index can be a negative value. A -1 refers to the first point of the linestring from the right, -2 refers to the second point of the linestring from the right, and so on.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be POINT.

## Return type

GEOMETRY

If *geom2* is the empty point, then *geom1* is returned.

If *geom1*, *geom2*, or *index* is null, then null is returned.

If *geom1* is not a linestring, then an error is returned.

If *index* is not within a valid index range, then an error is returned.

If *geom2* is not a point, then an error is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

## Examples

The following SQL returns a new linestring where we set the second point of the input linestring with the specified point.

```
SELECT ST_AsText(ST_SetPoint(ST_GeomFromText('LINESTRING(1 2, 3 2, 5 2, 1 2)'), 2,
ST_GeomFromText('POINT(7 9)')));
```

```
st_astext

LINESTRING(1 2,3 2,7 9,1 2)
```

The following SQL example returns a new linestring where we set the third point from the right (the index is negative) of the linestring with the specified point.

```
SELECT ST_AsText(ST_SetPoint(ST_GeomFromText('LINESTRING(1 2, 3 2, 5 2, 1 2)'), -3,
ST_GeomFromText('POINT(7 9)')));
```

```
st_astext

LINESTRING(1 2,7 9,5 2,1 2)
```

## ST\_SetSRID

ST\_SetSRID returns a geometry that is the same as input geometry, except updated with the value input for the spatial reference system identifier (SRID).

### Syntax

```
ST_SetSRID(geom, srnid)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*srid*

A value of data type INTEGER that represents a SRID.

## Return type

GEOMETRY

The SRID value of the returned geometry is set to *srid*.

If *geom* or *srid* is null, then null is returned.

If *srid* is negative, then an error is returned.

## Examples

The following SQL sets the SRID value of a linestring.

```
SELECT ST_AsEWKT(ST_SetSRID(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'),50));
```

```
st_asewkt
```

```

SRID=50;LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)
```

## ST\_Simplify

ST\_Simplify returns a simplified copy of the input geometry using the Ramer-Douglas-Peucker algorithm with the given tolerance. The topology of the input geometry might not be preserved. For more information about the algorithm, see [Ramer–Douglas–Peucker algorithm](#) in Wikipedia.

When ST\_Simplify calculates distances to simplify a geometry, ST\_Simplify operates on the 2D projection of the input geometry.

## Syntax

```
ST_Simplify(geom, tolerance)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*tolerance*

A value of data type DOUBLE PRECISION that represents the tolerance level of the Ramer-Douglas-Peucker algorithm. If *tolerance* is a negative number, then zero is used.

## Return type

GEOMETRY.

The spatial reference system identifier (SRID) value of the returned geometry is the SRID value of the input geometry.

The dimension of the returned geometry is the same as that of the input geometry.

If *geom* is null, then null is returned.

## Examples

The following SQL simplifies the input linestring using a Euclidean distance tolerance of 1 with the Ramer-Douglas-Peucker algorithm. The units of the distance are the same as those of the coordinates of the geometry.

```
SELECT ST_AsEWKT(ST_Simplify(ST_GeomFromText('LINESTRING(0 0,1 2,1 1,2 2,2 1)'), 1));
```

```
st_asewkt

LINESTRING(0 0,1 2,2 1)
```

## ST\_SRID

ST\_SRID returns the spatial reference system identifier (SRID) of an input geometry.

### Syntax

```
ST_SRID(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

INTEGER representing the SRID value of *geom*.

If *geom* is null, then null is returned.

## Examples

The following SQL returns an SRID value of a linestring.

```
SELECT ST_SRID(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
```

```
st_srid

0
```

## ST\_StartPoint

`ST_StartPoint` returns the first point of an input linestring. The spatial reference system identifier (SRID) value of the result is the same as that of the input geometry. The dimension of the returned geometry is the same as that of the input geometry.

### Syntax

```
ST_StartPoint(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. The subtype must be LINESTRING.

### Return type

GEOMETRY

If *geom* is null, then null is returned.

If *geom* is empty, then null is returned.

If *geom* isn't a LINESTRING, then null is returned.

### Examples

The following SQL returns an extended well-known text (EWKT) representation of a four-point LINESTRING to a GEOMETRY object and returns the start point of the linestring.

```
SELECT ST_AsEWKT(ST_StartPoint(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326)));
```

```
st_asewkt

SRID=4326;POINT(0 0)
```

## ST\_Touches

`ST_Touches` returns true if the 2D projections of the two input geometries touch. The two geometries touch if they are nonempty, intersect, and have no interior points in common.

### Syntax

```
ST_Touches(geom1, geom2)
```

### Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

BOOLEAN

If *geom1* or *geom2* is null, then null is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geom1* or *geom2* is a geometry collection, then an error is returned.

## Examples

The following SQL checks if a polygon touches a linestring.

```
SELECT ST_Touches(ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))'),
 ST_GeomFromText('LINESTRING(20 10,20 0,10 0)'));
```

```
st_touches

t
```

## ST\_Transform

ST\_Transform returns a new geometry with coordinates that are transformed in a spatial reference system defined by the input spatial reference system identifier (SRID).

## Syntax

```
ST_Transform(geom, srid)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*srid*

A value of data type INTEGER that represents an SRID.

## Return type

GEOMETRY.

The SRID value of the returned geometry is set to *srid*.

If *geom* or *srid* is null, then null is returned.

If the SRID value associated with the input *geom* does not exist, then an error is returned.

If *srid* does not exist, then an error is returned.

## Examples

The following SQL transforms the SRID of an empty geometry collection.

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('GEOMETRYCOLLECTION EMPTY', 3857), 4326));
```

```
st_asewkt

SRID=4326;GEOMETRYCOLLECTION EMPTY
```

The following SQL transforms the SRID of a linestring.

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('LINESTRING(110 40, 2 3, -10 80, -7 9, -22 -33)', 4326), 26918));
```

```
st_asewkt

SRID=26918;LINESTRING(73106.6977300955 15556182.9688576,14347201.5059964
1545178.32934967,1515090.41262989 9522193.25115316,10491250.83295
2575457.28410878,5672303.72135968 -5233682.61176205)
```

The following SQL transforms the SRID of a polygon.

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('POLYGON Z ((-10 10 -7, -65 10 -6, -10 64 -5,
-10 10 -7), (-11 11 5, -11 12 6, -12 11 7, -11 11 5))', 6989), 6317));
```

```
st_asewkt

SRID=6317;POLYGON Z ((6186430.2771091 -1090834.57212608 1100247.33216237,2654831.67853801
-5693304.90741276 1100247.50581055,2760987.41750022 -486836.575101877
5709710.44137268,6186430.2771091 -1090834.57212608 1100247.33216237),
(6146675.25029258 -1194792.63532103 1209007.1115113,6125027.87562215 -1190584.81194058
1317403.77865723,6124888.99555252 -1301885.3455052 1209007.49312929,6146675.25029258
-1194792.63532103 1209007.1115113))
```

## ST\_Union

`ST_Union` returns a geometry representing the union of two geometries. That is, it merges the input geometries to produce a resulting geometry with no overlaps.

### Syntax

```
ST_Union(geom1, geom2)
```

### Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

GEOMETRY

The spatial reference system identifier (SRID) value of the returned geometry is the SRID value of the input geometries.

If *geom1* or *geom2* is null, then null is returned.

If *geom1* or *geom2* are empty, then an empty geometry is returned.

If *geom1* and *geom2* don't have the same value for the spatial reference system identifier (SRID), then an error is returned.

If *geom1* or *geom2* is a geometry collection, linestring, or multilinestring, then an error is returned.

If *geom1* or *geom2* is not a two-dimensional (2D) geometry, then an error is returned.

## Examples

The following SQL returns the non-empty geometry representing the union of two input geometries.

```
SELECT ST_AsEWKT(ST_Union(ST_GeomFromText('POLYGON((0 0,100 100,0 200,0 0))'),
ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0)')));
```

```
st_asewkt
```

```
POLYGON((0 0,0 200,100 100,5 5,10 0,0 0))
```

## ST\_Within

ST\_Within returns true if the 2D projection of the first input geometry is within the 2D projection of the second input geometry.

For example, geometry A is within geometry B if every point in A is a point in B and their interiors have nonempty intersection.

ST\_Within(A, B) is equivalent to ST\_Contains(B, A).

## Syntax

```
ST_Within(geom1, geom2)
```

## Arguments

*geom1*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type. This value is compared with *geom2* to determine if it is within *geom2*.

*geom2*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

BOOLEAN

If *geom1* or *geom2* is null, then null is returned.

If *geom1* and *geom2* don't have the same spatial reference system identifier (SRID) value, then an error is returned.

If *geom1* or *geom2* is a geometry collection, then an error is returned.

## Examples

The following SQL checks if the first polygon is within the second polygon.

```
SELECT ST_Within(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_within

true
```

## ST\_X

ST\_X returns the first coordinate of an input point.

## Syntax

```
ST_X(point)
```

## Arguments

*point*

A POINT value of data type GEOMETRY.

## Return type

DOUBLE PRECISION value of the first coordinate.

If *point* is null, then null is returned.

If *point* is the empty point, then null is returned.

If *point* is not a POINT, then an error is returned.

## Examples

The following SQL returns the first coordinate of a point.

```
SELECT ST_X(ST_Point(1,2));
```

```
st_x

```

1.0

## ST\_XMax

ST\_XMax returns the maximum first coordinate of an input geometry.

### Syntax

```
ST_XMax(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

DOUBLE PRECISION value of the maximum first coordinate.

If *geom* is empty, then null is returned.

If *geom* is null, then null is returned.

### Examples

The following SQL returns the largest first coordinate of a linestring.

```
SELECT ST_XMax(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
```

```
st_xmax

77.42
```

## ST\_XMin

ST\_XMin returns the minimum first coordinate of an input geometry.

### Syntax

```
ST_XMin(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

DOUBLE PRECISION value of the minimum first coordinate.

If *geom* is empty, then null is returned.

If *geom* is null, then null is returned.

## Examples

The following SQL returns the smallest first coordinate of a linestring.

```
SELECT ST_XMin(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
```

```
st_xmin

77.27
```

## ST\_Y

ST\_Y returns the second coordinate of an input point.

### Syntax

```
ST_Y(point)
```

### Arguments

*point*

A POINT value of data type GEOMETRY.

### Return type

DOUBLE PRECISION value of the second coordinate.

If *point* is null, then null is returned.

If *point* is the empty point, then null is returned.

If *point* is not a POINT, then an error is returned.

## Examples

The following SQL returns the second coordinate of a point.

```
SELECT ST_Y(ST_Point(1,2));
```

```
st_y

2.0
```

## ST\_YMax

ST\_YMax returns the maximum second coordinate of an input geometry.

## Syntax

```
ST_YMax(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

DOUBLE PRECISION value of the maximum second coordinate.

If *geom* is empty, then null is returned.

If *geom* is null, then null is returned.

## Examples

The following SQL returns the largest second coordinate of a linestring.

```
SELECT ST_YMax(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
```

```
st_ymax

29.31
```

## ST\_YMin

ST\_YMin returns the minimum second coordinate of an input geometry.

## Syntax

```
ST_YMin(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

DOUBLE PRECISION value of the minimum second coordinate.

If *geom* is empty, then null is returned.

If *geom* is null, then null is returned.

## Examples

The following SQL returns the smallest second coordinate of a linestring.

```
SELECT ST_YMin(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));
```

```
st_ymin

29.07
```

## ST\_Z

ST\_Z returns the z coordinate of an input point.

### Syntax

```
ST_Z(point)
```

### Arguments

*point*

A POINT value of data type GEOMETRY.

### Return type

DOUBLE PRECISION value of the m coordinate.

If *point* is null, then null is returned.

If *point* is a 2D or 3DM point, then null is returned.

If *point* is the empty point, then null is returned.

If *point* is not a POINT, then an error is returned.

### Examples

The following SQL returns the z coordinate of a point in a 3DZ geometry.

```
SELECT ST_Z(ST_GeomFromEWKT('POINT Z (1 2 3)'));
```

```
st_z

3
```

The following SQL returns the z coordinate of a point in a 4D geometry.

```
SELECT ST_Z(ST_GeomFromEWKT('POINT ZM (1 2 3 4)'));
```

```
st_z

```

3

## ST\_ZMax

ST\_ZMax returns the maximum z coordinate of an input geometry.

### Syntax

```
ST_ZMax(geom)
```

### Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

### Return type

DOUBLE PRECISION value of the maximum z coordinate.

If *geom* is empty, then null is returned.

If *geom* is null, then null is returned.

If *geom* is a 2D or 3DM geometry, then null is returned.

### Examples

The following SQL returns the largest z coordinate of a linestring in a 3DZ geometry.

```
SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING Z (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_zmax

8
```

The following SQL returns the largest z coordinate of a linestring in a 4D geometry.

```
SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_zmax

10
```

## ST\_ZMin

ST\_ZMin returns the minimum z coordinate of an input geometry.

### Syntax

```
ST_ZMin(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

DOUBLE PRECISION value of the minimum z coordinate.

If *geom* is empty, then null is returned.

If *geom* is null, then null is returned.

If *geom* is a 2D or 3DM geometry, then null is returned.

## Examples

The following SQL returns the smallest z coordinate of a linestring in a 3DZ geometry.

```
SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING Z (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_zmin

2
```

The following SQL returns the smallest z coordinate of a linestring in a 4D geometry.

```
SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_zmin

2
```

## SupportsBBox

SupportsBBox returns true if the input geometry supports encoding with a precomputed bounding box. For more information about support for bounding boxes, see [Bounding box \(p. 229\)](#).

## Syntax

```
SupportsBBox(geom)
```

## Arguments

*geom*

A value of data type GEOMETRY or an expression that evaluates to a GEOMETRY type.

## Return type

BOOLEAN

If *geom* is null, then null is returned.

## Examples

The following SQL returns true because the input point geometry supports being encoded with a bounding box.

```
SELECT SupportsBBox(AddBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
supportsbbox

t
```

The following SQL returns false because the input point geometry doesn't support being encoded with a bounding box.

```
SELECT SupportsBBox(DropBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
supportsbbox

f
```

# String functions

## Topics

- [|| \(Concatenation\) operator \(p. 1278\)](#)
- [ASCII function \(p. 1280\)](#)
- [BPCHARCMP function \(p. 1280\)](#)
- [BTRIM function \(p. 1282\)](#)
- [BTTEXT\\_PATTERN\\_CMP function \(p. 1283\)](#)
- [CHAR\\_LENGTH function \(p. 1283\)](#)
- [CHARACTER\\_LENGTH function \(p. 1283\)](#)
- [CHARINDEX function \(p. 1283\)](#)
- [CHR function \(p. 1284\)](#)
- [COLLATE function \(p. 1285\)](#)
- [CONCAT function \(p. 1286\)](#)
- [CRC32 function \(p. 1288\)](#)
- [DIFFERENCE function \(p. 1288\)](#)
- [INITCAP function \(p. 1289\)](#)
- [LEFT and RIGHT functions \(p. 1291\)](#)
- [LEN function \(p. 1292\)](#)
- [LENGTH function \(p. 1293\)](#)
- [LOWER function \(p. 1293\)](#)
- [LPAD and RPAD functions \(p. 1294\)](#)
- [LTRIM function \(p. 1295\)](#)
- [OCTETINDEX function \(p. 1297\)](#)

- [OCTET\\_LENGTH function \(p. 1297\)](#)
- [POSITION function \(p. 1298\)](#)
- [QUOTE\\_IDENT function \(p. 1299\)](#)
- [QUOTE\\_LITERAL function \(p. 1300\)](#)
- [REGEXP\\_COUNT function \(p. 1301\)](#)
- [REGEXP\\_INSTR function \(p. 1302\)](#)
- [REGEXP\\_REPLACE function \(p. 1304\)](#)
- [REGEXP\\_SUBSTR function \(p. 1306\)](#)
- [REPEAT function \(p. 1309\)](#)
- [REPLACE function \(p. 1309\)](#)
- [REPLICATE function \(p. 1310\)](#)
- [REVERSE function \(p. 1311\)](#)
- [RTRIM function \(p. 1311\)](#)
- [SOUNDEX function \(p. 1313\)](#)
- [SPLIT\\_PART function \(p. 1314\)](#)
- [STRPOS function \(p. 1315\)](#)
- [strtol function \(p. 1317\)](#)
- [SUBSTR function \(p. 1318\)](#)
- [SUBSTRING function \(p. 1318\)](#)
- [TEXTLEN function \(p. 1321\)](#)
- [TRANSLATE function \(p. 1321\)](#)
- [TRIM function \(p. 1323\)](#)
- [UPPER function \(p. 1324\)](#)

String functions process and manipulate character strings or expressions that evaluate to character strings. When the *string* argument in these functions is a literal value, it must be enclosed in single quotation marks. Supported data types include CHAR and VARCHAR.

The following section provides the function names, syntax, and descriptions for supported functions. All offsets into strings are one-based.

#### Deprecated leader node-only functions

The following string functions are deprecated because they run only on the leader node. For more information, see [Leader node-only functions \(p. 1020\)](#)

- GET\_BYTE
- SET\_BIT
- SET\_BYTE
- TO\_ASCII

## || (Concatenation) operator

Concatenates two expressions on either side of the || symbol and returns the concatenated expression.

Similar to [CONCAT function \(p. 1286\)](#).

#### Note

For both the CONCAT function and the concatenation operator, if one or both expressions is null, the result of the concatenation is null.

## Syntax

```
expression1 || expression2
```

## Arguments

*expression1, expression2*

Both arguments can be fixed-length or variable-length character strings or expressions.

## Return type

The `||` operator returns a string. The type of string is the same as the input arguments.

## Example

The following example concatenates the FIRSTNAME and LASTNAME fields from the USERS table:

```
select firstname || ' ' || lastname
from users
order by 1
limit 10;

?column?

Aaron Banks
Aaron Booth
Aaron Browning
Aaron Burnett
Aaron Casey
Aaron Cash
Aaron Castro
Aaron Dickerson
Aaron Dixon
Aaron Dotson
(10 rows)
```

To concatenate columns that might contain nulls, use the [NVL function \(p. 1061\)](#) expression. The following example uses NVL to return a 0 whenever NULL is encountered.

```
select venuename || ' seats ' || nvl(venueseats, 0)
from venue where venuestate = 'NV' or venuestate = 'NC'
order by 1
limit 10;

seating

Ballys Hotel seats 0
Bank of America Stadium seats 73298
Bellagio Hotel seats 0
Caesars Palace seats 0
Harrahs Hotel seats 0
Hilton Hotel seats 0
Luxor Hotel seats 0
Mandalay Bay Hotel seats 0
Mirage Hotel seats 0
New York New York seats 0
```

## ASCII function

The ASCII function returns the ASCII code, or the Unicode code-point, of the first character in the string that you specify.

### Syntax

```
ASCII(string)
```

### Argument

*string*

You specify a CHAR or VARCHAR string for whose first character you want the ASCII code or Unicode code-point.

### Return type

The ASCII function returns an INTEGER value with the ASCII code or Unicode code-point of the first character of the string that you specify. The function returns 0 if the string is empty. It returns null if the string is null.

### Examples

The following example returns the ASCII code 97 for the first letter of the word amazon.

```
select ascii('amazon');
```

This example returns the following output.

```
ascii

97
```

The following example returns the ASCII code 65 for the first letter of the word Amazon.

```
select ascii('Amazon');
```

This example returns the following output.

```
ascii

65
```

## BPCHARCMP function

Compares the value of two strings and returns an integer. If the strings are identical, returns 0. If the first string is "greater" alphabetically, returns 1. If the second string is "greater", returns -1.

For multibyte characters, the comparison is based on the byte encoding.

Synonym of [BTTEXT\\_PATTERN\\_CMP function \(p. 1283\)](#).

## Syntax

```
BPCHARCMP(string1, string2)
```

## Arguments

*string1*

The first input parameter is a CHAR or VARCHAR string.

*string2*

The second parameter is a CHAR or VARCHAR string.

## Return type

The BPCHARCMP function returns an integer.

## Examples

The following example determines whether a user's first name is alphabetically greater than the user's last name for the first ten entries in USERS:

```
select userid, firstname, lastname,
bpcharcmp(firstname, lastname)
from users
order by 1, 2, 3, 4
limit 10;
```

This example returns the following sample output:

| userid | firstname | lastname  | bpcharcmp |
|--------|-----------|-----------|-----------|
| 1      | Rafael    | Taylor    | -1        |
| 2      | Vladimir  | Humphrey  | 1         |
| 3      | Lars      | Ratliff   | -1        |
| 4      | Barry     | Roy       | -1        |
| 5      | Reagan    | Hodge     | 1         |
| 6      | Victor    | Hernandez | 1         |
| 7      | Tamekah   | Juarez    | 1         |
| 8      | Colton    | Roy       | -1        |
| 9      | Mufutau   | Watkins   | -1        |
| 10     | Naida     | Calderon  | 1         |

(10 rows)

You can see that for entries where the string for the FIRSTNAME is later alphabetically than the LASTNAME, BPCHARCMP returns 1. If the LASTNAME is alphabetically later than FIRSTNAME, BPCHARCMP returns -1.

This example returns all entries in the USER table whose FIRSTNAME is identical to their LASTNAME:

```
select userid, firstname, lastname,
bpcharcmp(firstname, lastname)
from users where bpcharcmp(firstname, lastname)=0
order by 1, 2, 3, 4;
```

```
userid | firstname | lastname | bpcharcmp
```

| 62       | Chase   | Chase   |  | 0 |
|----------|---------|---------|--|---|
| 4008     | Whitney | Whitney |  | 0 |
| 12516    | Graham  | Graham  |  | 0 |
| 13570    | Harper  | Harper  |  | 0 |
| 16712    | Cooper  | Cooper  |  | 0 |
| 18359    | Chase   | Chase   |  | 0 |
| 27530    | Bradley | Bradley |  | 0 |
| 31204    | Harding | Harding |  | 0 |
| (8 rows) |         |         |  |   |

## BTRIM function

The BTRIM function trims a string by removing leading and trailing blanks or by removing leading and trailing characters that match an optional specified string.

### Syntax

```
BTRIM(string [, trim_chars])
```

### Arguments

*string*

The input VARCHAR string to be trimmed.

*trim\_chars*

The VARCHAR string containing the characters to be matched.

### Return type

The BTRIM function returns a VARCHAR string.

### Examples

The following example trims leading and trailing blanks from the string ' abc ':

```
select ' abc ' as untrim, btrim(' abc ') as trim;
untrim | trim
-----+-----
abc | abc
```

The following example removes the leading and trailing 'xyz' strings from the string 'xyzaxyzbxyzcxyz'. The leading and trailing occurrences of 'xyz' are removed, but occurrences that are internal within the string are not removed.

```
select 'xyzaxyzbxyzcxyz' as untrim,
btrim('xyzaxyzbxyzcxyz', 'xyz') as trim;
untrim | trim
-----+-----
xyzaxyzbxyzcxyz | axyzbxyzc
```

The following example removes the leading and trailing parts from the string 'setuphistorycassettes' that match any of the characters in the *trim\_chars* list 'tes'. Any t, e, or

s that occur before another character that is not in the *trim\_chars* list at the beginning or ending of the input string are removed.

```
SELECT btrim('setuphistorycassettes', 'tes');

btrim

uphistoryca
```

## BTTEXT\_PATTERN\_CMP function

Synonym for the BPCHARCMP function.

See [BPCHARCMP function \(p. 1280\)](#) for details.

## CHAR\_LENGTH function

Synonym of the LEN function.

See [LEN function \(p. 1292\)](#).

## CHARACTER\_LENGTH function

Synonym of the LEN function.

See [LEN function \(p. 1292\)](#).

## CHARINDEX function

Returns the location of the specified substring within a string. Synonym of the STRPOS function.

### Syntax

```
CHARINDEX(substring, string)
```

### Arguments

*substring*

The substring to search for within the *string*.

*string*

The string or column to be searched.

### Return type

The CHARINDEX function returns an integer corresponding to the position of the substring (one-based, not zero-based). The position is based on the number of characters, not bytes, so that multi-byte characters are counted as single characters.

### Usage notes

CHARINDEX returns 0 if the substring is not found within the *string*:

```
select charindex('dog', 'fish');
```

```
charindex

0
(1 row)
```

## Examples

The following example shows the position of the string fish within the word dogfish:

```
select charindex('fish', 'dogfish');
charindex

4
(1 row)
```

The following example returns the number of sales transactions with a COMMISSION over 999.00 from the SALES table:

```
select distinct charindex('.', commission), count (charindex('.', commission))
from sales where charindex('.', commission) > 4 group by charindex('.', commission)
order by 1,2;

charindex | count
-----+-----
5 | 629
(1 row)
```

See [STRPOS function \(p. 1315\)](#) for details.

## CHR function

The CHR function returns the character that matches the ASCII code point value specified by of the input parameter.

### Syntax

```
CHR(number)
```

### Argument

*number*

The input parameter is an integer that represents an ASCII code point value.

### Return type

The CHR function returns a CHAR string if an ASCII character matches the input value. If the input number has no ASCII match, the function returns null.

### Example

The following example returns event names that begin with a capital A (ASCII code point 65):

```
select distinct eventname from event
where substring(eventname, 1, 1)=chr(65);
```

```
eventname

Adriana Lecouvreur
A Man For All Seasons
A Bronx Tale
A Christmas Carol
Allman Brothers Band
...
```

## COLLATE function

The COLLATE function overrides the collation of a string column or expression.

For information on how to create tables using database collation, see [CREATE TABLE \(p. 823\)](#).

For information on how to create databases using database collation, see [CREATE DATABASE \(p. 745\)](#).

### Syntax

```
COLLATE(string, 'case_sensitive' | 'case_insensitive');
```

### Argument

*string*

The COLLATE function takes two input arguments. The first input is a string column or expression that you want to override. The second input is a string constant of a collation name. Amazon Redshift only supports *case\_sensitive* or *case\_insensitive*.

### Return type

The COLLATE function returns VARCHAR or CHAR depending on the first input expression type. This function only changes the collation of the first input argument and won't change its output value.

### Example

The following example defines col1 in table T as *case\_sensitive*. When you run the first query, Amazon Redshift only returns one row 'john'. After collate function runs on col1, the collation becomes *case\_insensitive*. The second query returns both 'john' and 'JOHN'.

```
CREATE TABLE T (
 col1 Varchar(20) COLLATE case_sensitive
);
```

```
INSERT INTO T VALUES ('john'),('JOHN');
```

```
SELECT * FROM T WHERE col1 = 'john';
col1

john
(1 rows)
```

```
SELECT * FROM T WHERE collate(col1, 'case_insensitive') = 'john';
col1
```

```

john
JOHN
(2 rows)
```

## CONCAT function

The CONCAT function concatenates two expressions and returns the resulting expression. To concatenate more than two expressions, use nested CONCAT functions. The concatenation operator (||) between two expressions produces the same results as the CONCAT function.

### Note

For both the CONCAT function and the concatenation operator, if one or both expressions is null, the result of the concatenation is null.

## Syntax

```
CONCAT (expression1, expression2)
```

## Arguments

*expression1*, *expression2*

Both arguments can be a fixed-length character string, a variable-length character string, a binary expression, or an expression that evaluates to one of these inputs.

## Return type

CONCAT returns an expression. The data type of the expression is the same type as the input arguments.

If the input expressions are of different types, Amazon Redshift tries to implicitly type casts one of the expressions. If values can't be cast, an error is returned.

## Examples

The following example concatenates two character literals:

```
select concat('December 25, ', '2008');

concat

December 25, 2008
(1 row)
```

The following query, using the || operator instead of CONCAT, produces the same result:

```
select 'December 25, '||'2008';

?column?

December 25, 2008
(1 row)
```

The following example uses two CONCAT functions to concatenate three character strings:

```
select concat('Thursday, ', concat('December 25, ', '2008'));
```

```
concat

Thursday, December 25, 2008
(1 row)
```

To concatenate columns that might contain nulls, use the [NVL function \(p. 1061\)](#). The following example uses NVL to return a 0 whenever NULL is encountered.

```
select concat(venuename, concat(' seats ', nvl(venueseats, 0))) as seating
from venue where venuestate = 'NV' or venuestate = 'NC'
order by 1
limit 5;

seating

Ballys Hotel seats 0
Bank of America Stadium seats 73298
Bellagio Hotel seats 0
Caesars Palace seats 0
Harrahs Hotel seats 0
(5 rows)
```

The following query concatenates CITY and STATE values from the VENUE table:

```
select concat(venuecity, venuestate)
from venue
where venueseats > 75000
order by venueseats;

concat

DenverCO
Kansas CityMO
East RutherfordNJ
LandoverMD
(4 rows)
```

The following query uses nested CONCAT functions. The query concatenates CITY and STATE values from the VENUE table but delimits the resulting string with a comma and a space:

```
select concat(concat(venuecity, ','), venuestate)
from venue
where venueseats > 75000
order by venueseats;

concat

Denver, CO
Kansas City, MO
East Rutherford, NJ
Landover, MD
(4 rows)
```

The following example concatenates two binary expressions. Where abc is a binary value (with a hexadecimal representation of 616263) and def is a binary value (with hexadecimal representation of 646566). The result is automatically shown as the hexadecimal representation of the binary value.

```
select concat('abc'::varbyte, 'def'::varbyte);
```

```
concat
```

```

```

```
616263646566
```

## CRC32 function

CRC32 is an error-detecting function that uses a CRC32 algorithm to detect changes between source and target data. The CRC32 function converts a variable-length string into an 8-character string that is a text representation of the hexadecimal value of a 32 bit-binary sequence.

### Syntax

```
CRC32(string)
```

### Arguments

*string*

A variable-length string.

### Return type

The CRC32 function returns an 8-character string that is a text representation of the hexadecimal value of a 32-bit binary sequence. The Amazon Redshift CRC32 function is based on the CRC-32C polynomial.

### Example

The following example shows the 32-bit value for the string 'Amazon Redshift':

```
select crc32('Amazon Redshift');
crc32

f2726906
(1 row)
```

## DIFFERENCE function

The DIFFERENCE function compares two strings by converting the strings to American Soundex codes and returning an INTEGER to indicate the difference between the codes.

### Syntax

```
DIFFERENCE(string1, string2)
```

### Arguments

*string1*

You specify the first CHAR or VARCHAR string of two strings that you want the function to compare.

*string2*

You specify the second CHAR or VARCHAR string of two strings that you want the function to compare.

## Return type

The DIFFERENCE function returns an INTEGER value 0–4 that counts the number of different letters in the two strings compared as American Soundex codes. The function returns 4 if the American Soundex code value is the same.

## Usage notes

The DIFFERENCE function converts only English alphabetical lowercase or uppercase ASCII characters, including a-z and A-Z. DIFFERENCE ignores other characters. DIFFERENCE returns 0 if one of the two strings doesn't contain valid characters. It returns 1 if neither string contains valid characters.

```
select difference('Amazon', '+-*%');
```

```
difference

0
```

## Example

The following example returns 4 because the Soundex value for the two strings is the same.

```
select difference('AC/DC', 'Ay See Dee See');
```

```
difference

4
```

## INITCAP function

Capitalizes the first letter of each word in a specified string. INITCAP supports UTF-8 multibyte characters, up to a maximum of four bytes per character.

## Syntax

```
INITCAP(string)
```

## Argument

*string*

The input parameter is a CHAR or VARCHAR string.

## Return type

The INITCAP function returns a VARCHAR string.

## Usage notes

The INITCAP function makes the first letter of each word in a string uppercase, and any subsequent letters are made (or left) lowercase. Therefore, it is important to understand which characters (other than space characters) function as word separators. A *word separator* character is any non-alphanumeric

character, including punctuation marks, symbols, and control characters. All of the following characters are word separators:

```
! " # $ % & ' () * + , - . / : ; < = > ? @ [\] ^ _ ` { | } ~
```

Tabs, newline characters, form feeds, line feeds, and carriage returns are also word separators.

## Examples

The following example capitalizes the initials of each word in the CATDESC column:

| catid     | catdesc                                    | initcap                                    |
|-----------|--------------------------------------------|--------------------------------------------|
| 1         | Major League Baseball                      | Major League Baseball                      |
| 2         | National Hockey League                     | National Hockey League                     |
| 3         | National Football League                   | National Football League                   |
| 4         | National Basketball Association            | National Basketball Association            |
| 5         | Major League Soccer                        | Major League Soccer                        |
| 6         | Musical theatre                            | Musical Theatre                            |
| 7         | All non-musical theatre                    | All Non-Musical Theatre                    |
| 8         | All opera and light opera                  | All Opera And Light Opera                  |
| 9         | All rock and pop music concerts            | All Rock And Pop Music Concerts            |
| 10        | All jazz singers and bands                 | All Jazz Singers And Bands                 |
| 11        | All symphony, concerto, and choir concerts | All Symphony, Concerto, And Choir Concerts |
| Concerts  |                                            |                                            |
| (11 rows) |                                            |                                            |

The following example shows that the INITCAP function does not preserve uppercase characters when they do not begin words. For example, MLB becomes Mlb.

| initcap   |
|-----------|
| Classical |
| Jazz      |
| Mlb       |
| Mls       |
| Musicals  |
| Nba       |
| Nfl       |
| Nhl       |
| Opera     |
| Plays     |
| Pop       |
| (11 rows) |

The following example shows that non-alphanumeric characters other than spaces function as word separators, causing uppercase characters to be applied to several letters in each string:

```
select email, initcap(email)
from users
order by userid desc limit 5;
```

```

email | initcap

urna.Ut@egedictumplacerat.edu | Urna.Ut@Egedictumplacerat.Edu
nibh.enim@egestas.ca | Nibh.Enim@Egestas.Ca
in@Donecat.ca | In@Donecat.Ca
sodales@blanditviverraDonec.ca | Sodales@Blanditviverradonec.Ca
sociis.natoque.penatibus@vitae.org | Sociis.Natoque.Penatibus@Vitae.Org
(5 rows)

```

## LEFT and RIGHT functions

These functions return the specified number of leftmost or rightmost characters from a character string.

The number is based on the number of characters, not bytes, so that multibyte characters are counted as single characters.

### Syntax

|                                                      |
|------------------------------------------------------|
| <code>LEFT ( <i>string</i>, <i>integer</i> )</code>  |
| <code>RIGHT ( <i>string</i>, <i>integer</i> )</code> |

### Arguments

*string*

Any character string or any expression that evaluates to a character string.

*integer*

A positive integer.

### Return type

LEFT and RIGHT return a VARCHAR string.

### Example

The following example returns the leftmost 5 and rightmost 5 characters from event names that have IDs between 1000 and 1005:

```

select eventid, eventname,
left(eventname,5) as left_5,
right(eventname,5) as right_5
from event
where eventid between 1000 and 1005
order by 1;

eventid | eventname | left_5 | right_5
-----+-----+-----+-----+
 1000 | Gypsy | Gypsy | Gypsy
 1001 | Chicago | Chica | icago
 1002 | The King and I | The K | and I
 1003 | Pal Joey | Pal J | Joey
 1004 | Grease | Greas | rease
 1005 | Chicago | Chica | icago
(6 rows)

```

## LEN function

Returns the length of the specified string as the number of characters.

### Syntax

LEN is a synonym of [LENGTH function \(p. 1293\)](#), [CHAR\\_LENGTH function \(p. 1283\)](#), [CHARACTER\\_LENGTH function \(p. 1283\)](#), and [TEXTLEN function \(p. 1321\)](#).

```
LEN(expression)
```

### Argument

*expression*

The input parameter is a CHAR, VARCHAR, or VARBYTE or an alias of one of the valid input types.

### Return type

The LEN function returns an integer indicating the number of characters in the input string.

If the input string is a character string, the LEN function returns the actual number of characters in multi-byte strings, not the number of bytes. For example, a VARCHAR(12) column is required to store three four-byte Chinese characters. The LEN function will return 3 for that same string. To get the length of a string in bytes, use the [OCTET\\_LENGTH \(p. 1297\)](#) function.

### Usage notes

Length calculations do not count trailing spaces for fixed-length character strings but do count them for variable-length strings.

### Example

The following example returns the number of bytes and the number of characters in the string `français`.

```
select octet_length('français'),
len('français');

octet_length | len
-----+-----
 9 | 8
```

The following example returns the number of characters in the strings `cat` with no trailing spaces and `cat`  with three trailing spaces:

```
select len('cat'), len('cat ');\nlen | len\n-----+-----\n 3 | 6
```

The following example returns the ten longest VENUENAME entries in the VENUE table:

```
select venuename, len(venuename)
```

```
from venue
order by 2 desc, 1
limit 10;

venuename | len

Saratoga Springs Performing Arts Center | 39
Lincoln Center for the Performing Arts | 38
Nassau Veterans Memorial Coliseum | 33
Jacksonville Municipal Stadium | 30
Rangers BallPark in Arlington | 29
University of Phoenix Stadium | 29
Circle in the Square Theatre | 28
Hubert H. Humphrey Metrodome | 28
Oriole Park at Camden Yards | 27
Dick's Sporting Goods Park | 26
```

The following example returns the number of bytes of the binary value aaa.

```
select len('aaa'::varbyte);

len

3
```

## LENGTH function

Synonym of the LEN function.

See [LEN function \(p. 1292\)](#).

## LOWER function

Converts a string to lowercase. LOWER supports UTF-8 multibyte characters, up to a maximum of four bytes per character.

### Syntax

```
LOWER(string)
```

### Argument

*string*

The input parameter is a VARCHAR string (or any other data type, such as CHAR, that can be implicitly converted to VARCHAR).

### Return type

The LOWER function returns a character string that is the same data type as the input string.

### Examples

The following example converts the CATNAME field to lowercase:

```
select catname, lower(catname) from category order by 1,2;
```

```

catname | lower
-----+-----
Classical | classical
Jazz | jazz
MLB | mlb
MLS | mls
Musicals | musicals
NBA | nba
NFL | nfl
NHL | nhl
Opera | opera
Plays | plays
Pop | pop
(11 rows)

```

## LPAD and RPAD functions

These functions prepend or append characters to a string, based on a specified length.

### Syntax

|                                                  |
|--------------------------------------------------|
| <code>LPAD (string1, length, [ string2 ])</code> |
|--------------------------------------------------|

|                                                  |
|--------------------------------------------------|
| <code>RPAD (string1, length, [ string2 ])</code> |
|--------------------------------------------------|

### Arguments

#### *string1*

A character string or an expression that evaluates to a character string, such as the name of a character column.

#### *length*

An integer that defines the length of the result of the function. The length of a string is based on the number of characters, not bytes, so that multi-byte characters are counted as single characters. If *string1* is longer than the specified length, it is truncated (on the right). If *length* is a negative number, the result of the function is an empty string.

#### *string2*

One or more characters that are prepended or appended to *string1*. This argument is optional; if it is not specified, spaces are used.

### Return type

These functions return a VARCHAR data type.

### Examples

Truncate a specified set of event names to 20 characters and prepend the shorter names with spaces:

|                                                                                             |
|---------------------------------------------------------------------------------------------|
| <pre> select lpad(eventname,20) from event where eventid between 1 and 5 order by 1; </pre> |
|---------------------------------------------------------------------------------------------|

lpad

```

 Salome
 Il Trovatore
 Boris Godunov
 Gotterdamerung
 La Cenerentola (Cind
(5 rows)
```

Truncate the same set of event names to 20 characters but append the shorter names with 0123456789.

```
select rpad(eventname,20,'0123456789') from event
where eventid between 1 and 5 order by 1;

rpad

Boris Godunov0123456
Gotterdamerung01234
Il Trovatore01234567
La Cenerentola (Cind
Salome01234567890123
(5 rows)
```

## LTRIM function

Trims characters from the beginning of a string. Removes the longest string containing only characters in the trim characters list. Trimming is complete when a trim character does not appear in the input string.

### Syntax

```
LTRIM(string [, trim_chars])
```

### Arguments

*string*

A string column, expression, or string literal to be trimmed.

*trim\_chars*

A string column, expression, or string literal that represents the characters to be trimmed from the beginning of *string*. If not specified, a space is used as the trim character.

### Return type

The LTRIM function returns a character string that is the same data type as the input *string* (CHAR or VARCHAR).

### Examples

The following example trims the year from the *listtime* column. The trim characters in string literal '*2008-*' indicate the characters to be trimmed from the left. If you use the trim characters '*028-*', you achieve the same result.

```
select listid, listtime, ltrim(listtime, '2008-')
from listing
order by 1, 2, 3
```

```
limit 10;

listid | listtime | ltrim
-----+-----+
 1 | 2008-01-24 06:43:29 | 1-24 06:43:29
 2 | 2008-03-05 12:25:29 | 3-05 12:25:29
 3 | 2008-11-01 07:35:33 | 11-01 07:35:33
 4 | 2008-05-24 01:18:37 | 5-24 01:18:37
 5 | 2008-05-17 02:29:11 | 5-17 02:29:11
 6 | 2008-08-15 02:08:13 | 15 02:08:13
 7 | 2008-11-15 09:38:15 | 11-15 09:38:15
 8 | 2008-11-09 05:07:30 | 11-09 05:07:30
 9 | 2008-09-09 08:03:36 | 9-09 08:03:36
10 | 2008-06-17 09:44:54 | 6-17 09:44:54
```

LTRIM removes any of the characters in *trim\_chars* when they appear at the beginning of *string*. The following example trims the characters 'C', 'D', and 'G' when they appear at the beginning of VENUENAME, which is a VARCHAR column.

```
select venueid, venuename, ltrim(venuename, 'CDG')
from venue
where venuename like '%Park'
order by 2
limit 7;

venueid | venuename | btrim
-----+-----+
 121 | ATT Park | ATT Park
 109 | Citizens Bank Park | itizens Bank Park
 102 | Comerica Park | omerica Park
 9 | Dick's Sporting Goods Park | ick's Sporting Goods Park
 97 | Fenway Park | Fenway Park
 112 | Great American Ball Park | reat American Ball Park
 114 | Miller Park | Miller Park
```

The following example uses the trim character 2 which is retrieved from the venueid column.

```
select ltrim('2008-01-24 06:43:29', venueid)
from venue where venueid=2;

ltrim

008-01-24 06:43:29
```

The following example does not trim any characters because a 2 is found before the '0' trim character.

```
select ltrim('2008-01-24 06:43:29', '0');

ltrim

2008-01-24 06:43:29
```

The following example uses the default space trim character and trims the two spaces from the beginning of the string.

```
select ltrim(' 2008-01-24 06:43:29');

ltrim

```

2008-01-24 06:43:29

## OCTETINDEX function

The OCTETINDEX function returns the location of a substring within a string as a number of bytes.

### Syntax

```
OCTETINDEX(substring, string)
```

### Arguments

*substring*

You specify a CHAR or VARCHAR substring to locate within a string.

*string*

You specify a CHAR or VARCHAR string to search for the substring.

### Return type

The OCTETINDEX function returns an INTEGER value corresponding to the position of the substring within the string as a number of bytes, where the first character in the string is counted as 1. If the string doesn't contain multibyte characters, the result is equal to the result of the CHARINDEX function.

### Examples

The following example returns 8 because the substring AWS begins on the eighth byte of the string.

```
select octetindex('AWS', 'Amazon AWS');
```

```
octetindex

8
```

The following example returns 14 because the first six characters of the string are double-byte characters.

```
select octetindex('AWS', 'Αμαζον AWS');
```

```
octetindex

14
```

## OCTET\_LENGTH function

Returns the length of the specified string as the number of bytes.

### Syntax

```
OCTET_LENGTH(expression)
```

## Argument

*expression*

The input parameter is a CHAR, VARCHAR, or VARBYTE string.

## Return type

The OCTET\_LENGTH function returns an integer indicating the number of bytes in the input string.

If the input string is a character string, the [LEN \(p. 1292\)](#) function returns the actual number of characters in multi-byte strings, not the number of bytes. For example, to store three four-byte Chinese characters, you need a VARCHAR(12) column. The LEN function will return 3 for that same string.

## Usage notes

Length calculations do not count trailing spaces for fixed-length character strings but do count them for variable-length strings.

## Example

The following example returns the number of bytes and the number of characters in the string *français*.

```
select octet_length('français'),
len('français');

octet_length | len
-----+-----
 9 | 8
(1 row)
```

## POSITION function

Returns the location of the specified substring within a string.

Synonym of the [STRPOS function \(p. 1315\)](#) function.

## Syntax

```
POSITION(substring IN string)
```

## Arguments

*substring*

The substring to search for within the *string*.

*string*

The string or column to be searched.

## Return type

The POSITION function returns an integer corresponding to the position of the substring (one-based, not zero-based). The position is based on the number of characters, not bytes, so that multi-byte characters are counted as single characters.

## Usage notes

POSITION returns 0 if the substring is not found within the string:

```
select position('dog' in 'fish');

position

0
(1 row)
```

## Examples

The following example shows the position of the string fish within the word dogfish:

```
select position('fish' in 'dogfish');

position

4
(1 row)
```

The following example returns the number of sales transactions with a COMMISSION over 999.00 from the SALES table:

```
select distinct position('.' in commission), count (position('.' in commission))
from sales where position('.' in commission) > 4 group by position('.' in commission)
order by 1,2;

position | count
-----+-----
 5 | 629
(1 row)
```

## QUOTE\_IDENT function

The QUOTE\_IDENT function returns the specified string as a string in double quotation marks so that it can be used as an identifier in a SQL statement. Appropriately doubles any embedded double quotation marks.

QUOTE\_IDENT adds double quotation marks only where necessary to create a valid identifier, when the string contains non-identifier characters or would otherwise be folded to lowercase. To always return a single-quoted string, use [QUOTE\\_LITERAL \(p. 1300\)](#).

### Syntax

```
QUOTE_IDENT(string)
```

### Argument

*string*

The input parameter can be a CHAR or VARCHAR string.

### Return type

The QUOTE\_IDENT function returns the same type string as the input parameter.

## Example

The following example returns the CATNAME column surrounded by quotation marks.

```
select catid, quote_ident(catname)
from category
order by 1,2;

 catid | quote_ident
-----+-----
 1 | "MLB"
 2 | "NHL"
 3 | "NFL"
 4 | "NBA"
 5 | "MLS"
 6 | "Musicals"
 7 | "Plays"
 8 | "Opera"
 9 | "Pop"
 10 | "Jazz"
 11 | "Classical"
(11 rows)
```

## QUOTE\_LITERAL function

The QUOTE\_LITERAL function returns the specified string as a quoted string so that it can be used as a string literal in a SQL statement. If the input parameter is a number, QUOTE\_LITERAL treats it as a string. Appropriately doubles any embedded single quotation marks and backslashes.

### Syntax

```
QUOTE_LITERAL(string)
```

### Argument

*string*

The input parameter is a CHAR or VARCHAR string.

### Return type

The QUOTE\_LITERAL function returns a string that is the same data type as the input string (CHAR or VARCHAR).

## Example

The following example returns the CATID column surrounded by quotation marks. The ordering now treats this column as a string.

```
select quote_literal(catid), catname
from category
order by 1,2;

quote_literal | catname
-----+-----
'1' | MLB
'10' | Jazz
```

```
'11' | Classical
'2' | NHL
'3' | NFL
'4' | NBA
'5' | MLS
'6' | Musicals
'7' | Plays
'8' | Opera
'9' | Pop
(11 rows)
```

## REGEXP\_COUNT function

Searches a string for a regular expression pattern and returns an integer that indicates the number of times the pattern occurs in the string. If no match is found, then the function returns 0. For more information about regular expressions, see [POSIX operators \(p. 616\)](#).

### Syntax

```
REGEXP_COUNT (source_string, pattern [, position [, parameters]])
```

### Arguments

*source\_string*

A string expression, such as a column name, to be searched.

*pattern*

A string literal that represents a SQL standard regular expression pattern.

*position*

A positive integer that indicates the position within *source\_string* to begin searching. The position is based on the number of characters, not bytes, so that multibyte characters are counted as single characters. The default is 1. If *position* is less than 1, the search begins at the first character of *source\_string*. If *position* is greater than the number of characters in *source\_string*, the result is 0.

*parameters*

One or more string literals that indicate how the function matches the pattern. The possible values are the following:

- c – Perform case-sensitive matching. The default is to use case-sensitive matching.
- i – Perform case-insensitive matching.
- p – Interpret the pattern with Perl Compatible Regular Expression (PCRE) dialect.

### Return type

Integer

### Example

The following example counts the number of times a three-letter sequence occurs.

```
SELECT regexp_count('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');
regexp_count

```

The following example counts the number of times the top-level domain name is either org or edu.

```
SELECT email, regexp_count(email,'@[^.]*\\.(org|edu)')FROM users
ORDER BY userid LIMIT 4;
```

| email                                         | regexp_count |
|-----------------------------------------------|--------------|
| Etiam.laoreet.libero@sodalesMaurisblandit.edu | 1            |
| Suspendisse.tristique@nonnisiAenean.edu       | 1            |
| amet.faucibus.ut@condimentumegetvolutpat.ca   | 0            |
| sed@lacusUtneca.ca                            | 0            |

The following example counts the occurrences of the string FOX, using case-insensitive matching.

```
SELECT regexp_count('the fox', 'FOX', 1, 'i');
```

| regexp_count |
|--------------|
| 1            |

The following example uses a pattern written in the PCRE dialect to locate words containing at least one number and one lowercase letter. It uses the ?= operator, which has a specific look-ahead connotation in PCRE. This example counts the number of occurrences of such words, with case-sensitive matching.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^]*[a-z])(?=[^]*[0-9])[^]+', 1,
'p');
```

| regexp_count |
|--------------|
| 2            |

The following example uses a pattern written in the PCRE dialect to locate words containing at least one number and one lowercase letter. It uses the ?= operator, which has a specific connotation in PCRE. This example counts the number of occurrences of such words, but differs from the previous example in that it uses case-insensitive matching.

```
SELECT regexp_count('passwd7 plain A1234 a1234', '(?=[^]*[a-z])(?=[^]*[0-9])[^]+', 1,
'ip');
```

| regexp_count |
|--------------|
| 3            |

## REGEXP\_INSTR function

Searches a string for a regular expression pattern and returns an integer that indicates the beginning position or ending position of the matched substring. If no match is found, then the function returns 0. REGEXP\_INSTR is similar to the [POSITION \(p. 1298\)](#) function, but lets you search a string for a regular expression pattern. For more information about regular expressions, see [POSIX operators \(p. 616\)](#).

### Syntax

```
REGEXP_INSTR (source_string, pattern [, position [, occurrence] [, option [, parameters
]]])
```

## Arguments

*source\_string*

A string expression, such as a column name, to be searched.

*pattern*

A string literal that represents a SQL standard regular expression pattern.

*position*

A positive integer that indicates the position within *source\_string* to begin searching. The position is based on the number of characters, not bytes, so that multibyte characters are counted as single characters. The default is 1. If *position* is less than 1, the search begins at the first character of *source\_string*. If *position* is greater than the number of characters in *source\_string*, the result is 0.

*occurrence*

A positive integer that indicates which occurrence of the pattern to use. REGEXP\_INSTR skips the first *occurrence* - 1 matches. The default is 1. If *occurrence* is less than 1 or greater than the number of characters in *source\_string*, the search is ignored and the result is 0.

*option*

A value that indicates whether to return the position of the first character of the match (0) or the position of the first character following the end of the match (1). A nonzero value is the same as 1. The default value is 0.

*parameters*

One or more string literals that indicate how the function matches the pattern. The possible values are the following:

- c – Perform case-sensitive matching. The default is to use case-sensitive matching.
- i – Perform case-insensitive matching.
- e – Extract a substring using a subexpression.

If *pattern* includes a subexpression, REGEXP\_INSTR matches a substring using the first subexpression in *pattern*. REGEXP\_INSTR considers only the first subexpression; additional subexpressions are ignored. If the pattern doesn't have a subexpression, REGEXP\_INSTR ignores the 'e' parameter.

- p – Interpret the pattern with Perl Compatible Regular Expression (PCRE) dialect.

## Return type

Integer

## Example

The following example searches for the @ character that begins a domain name and returns the starting position of the first match.

```
SELECT email, regexp_instr(email, '@[^.]*)'
FROM users
ORDER BY userid LIMIT 4;

email | regexp_instr
-----+-----
Etiam.laoreet.libero@sodalesMaurisblandit.edu | 21
```

|                                             |  |    |
|---------------------------------------------|--|----|
| Suspendisse.tristique@nonnisiAenean.edu     |  | 22 |
| amet.faucibus.ut@condimentumegetvolutpat.ca |  | 17 |
| sed@lacusUtne.ca                            |  | 4  |

The following example searches for variants of the word Center and returns the starting position of the first match.

```
SELECT venueName, regexp_instr(venueName, '[cC]ent(er|re)$')
FROM venue
WHERE regexp_instr(venueName, '[cC]ent(er|re)$') > 0
ORDER BY venueId LIMIT 4;

venueName | regexp_instr
-----+-----
The Home Depot Center | 16
Izod Center | 6
Wachovia Center | 10
Air Canada Centre | 12
```

The following example finds the starting position of the first occurrence of the string FOX, using case-insensitive matching logic.

```
SELECT regexp_instr('the fox', 'FOX', 1, 1, 0, 'i');

regexp_instr

5
```

The following example uses a pattern written in PCRE dialect to locate words containing at least one number and one lowercase letter. It uses the ?= operator, which has a specific look-ahead connotation in PCRE. This example finds the starting position of the second such word.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^]*[a-z])(?=[^]*[0-9])[^]+', 1, 2,
0, 'p');

regexp_instr

21
```

The following example uses a pattern written in PCRE dialect to locate words containing at least one number and one lowercase letter. It uses the ?= operator, which has a specific look-ahead connotation in PCRE. This example finds the starting position of the second such word, but differs from the previous example in that it uses case-insensitive matching.

```
SELECT regexp_instr('passwd7 plain A1234 a1234', '(?=[^]*[a-z])(?=[^]*[0-9])[^]+', 1, 2,
0, 'ip');

regexp_instr

15
```

## REGEXP\_REPLACE function

Searches a string for a regular expression pattern and replaces every occurrence of the pattern with the specified string. REGEXP\_REPLACE is similar to the [REPLACE function \(p. 1309\)](#), but lets you search a string for a regular expression pattern. For more information about regular expressions, see [POSIX operators \(p. 616\)](#).

REGEXP\_REPLACE is similar to the [TRANSLATE function \(p. 1321\)](#) and the [REPLACE function \(p. 1309\)](#), except that TRANSLATE makes multiple single-character substitutions and REPLACE substitutes one entire string with another string, while REGEXP\_REPLACE lets you search a string for a regular expression pattern.

## Syntax

```
REGEXP_REPLACE (source_string, pattern [, replace_string [, position [, parameters]]])
```

## Arguments

*source\_string*

A string expression, such as a column name, to be searched.

*pattern*

A string literal that represents a SQL standard regular expression pattern.

*replace\_string*

A string expression, such as a column name, that will replace each occurrence of pattern. The default is an empty string ( "" ).

*position*

A positive integer that indicates the position within *source\_string* to begin searching. The position is based on the number of characters, not bytes, so that multibyte characters are counted as single characters. The default is 1. If *position* is less than 1, the search begins at the first character of *source\_string*. If *position* is greater than the number of characters in *source\_string*, the result is *source\_string*.

*parameters*

One or more string literals that indicate how the function matches the pattern. The possible values are the following:

- c – Perform case-sensitive matching. The default is to use case-sensitive matching.
- i – Perform case-insensitive matching.
- p – Interpret the pattern with Perl Compatible Regular Expression (PCRE) dialect.

## Return type

VARCHAR

If either *pattern* or *replace\_string* is NULL, the return is NULL.

## Example

The following example deletes the @ and domain name from email addresses.

```
SELECT email, regexp_replace(email, '@.*\\.(org|gov|com|edu|ca)$')
FROM users
ORDER BY userid LIMIT 4;

email | regexp_replace
-----+-----
Etiam.laoreet.libero@sodalesMaurisblandit.edu | Etiam.laoreet.libero
```

|                                                                                                                   |                                                           |
|-------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|
| <pre>Suspendisse.tristique@nonnisiAenean.edu amet.faucibus.ut@condimentumegetvolutpat.ca sed@lacusUtneca.ca</pre> | <pre>Suspendisse.tristique   amet.faucibus.ut   sed</pre> |
|-------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|

The following example replaces the domain names of email addresses with this value:  
`internal.company.com`.

|                                                                                                                                                                                                                                                                                                                                          |                                   |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| <pre>SELECT email, regexp_replace(email, '@.*\\.[[:alpha:]]{2,3}', '@internal.company.com') FROM users ORDER BY userid LIMIT 4;</pre>                                                                                                                                                                                                    | <pre>email   regexp_replace</pre> |
| <pre>-----+-----+ Etiam.laoreet.libero@sodalesMaurisblandit.edu   Etiam.laoreet.libero@internal.company.com Suspendisse.tristique@nonnisiAenean.edu   Suspendisse.tristique@internal.company.com amet.faucibus.ut@condimentumegetvolutpat.ca   amet.faucibus.ut@internal.company.com sed@lacusUtneca.ca   sed@internal.company.com</pre> |                                   |

The following example replaces all occurrences of the string FOX within the value quick brown fox, using case-insensitive matching.

|                                                                                |                           |
|--------------------------------------------------------------------------------|---------------------------|
| <pre>SELECT regexp_replace('the fox', 'FOX', 'quick brown fox', 1, 'i');</pre> | <pre>regexp_replace</pre> |
| <pre>-----+-----+ the quick brown fox</pre>                                    |                           |

The following example uses a pattern written in the PCRE dialect to locate words containing at least one number and one lowercase letter. It uses the `?=` operator, which has a specific look-ahead connotation in PCRE. This example replaces each occurrence of such a word with the value [hidden].

|                                                                                                                         |                           |
|-------------------------------------------------------------------------------------------------------------------------|---------------------------|
| <pre>SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+', '[hidden]', 1, 'p');</pre> | <pre>regexp_replace</pre> |
| <pre>-----+-----+ [hidden] plain A1234 [hidden]</pre>                                                                   |                           |

The following example uses a pattern written in the PCRE dialect to locate words containing at least one number and one lowercase letter. It uses the `?=` operator, which has a specific look-ahead connotation in PCRE. This example replaces each occurrence of such a word with the value [hidden], but differs from the previous example in that it uses case-insensitive matching.

|                                                                                                                          |                           |
|--------------------------------------------------------------------------------------------------------------------------|---------------------------|
| <pre>SELECT regexp_replace('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+', '[hidden]', 1, 'ip');</pre> | <pre>regexp_replace</pre> |
| <pre>-----+-----+ [hidden] plain [hidden] [hidden]</pre>                                                                 |                           |

## REGEXP\_SUBSTR function

Returns characters from a string by searching it for a regular expression pattern. REGEXP\_SUBSTR is similar to the [SUBSTRING function \(p. 1318\)](#) function, but lets you search a string for a regular expression pattern. If the function can't match the regular expression to any characters in the string, it returns an empty string. For more information about regular expressions, see [POSIX operators \(p. 616\)](#).

## Syntax

```
REGEXP_SUBSTR (source_string, pattern [, position [, occurrence [, parameters]]])
```

## Arguments

*source\_string*

A string expression to be searched.

*pattern*

A string literal that represents a SQL standard regular expression pattern.

*position*

A positive integer that indicates the position within *source\_string* to begin searching. The position is based on the number of characters, not bytes, so that multi-byte characters are counted as single characters. The default is 1. If *position* is less than 1, the search begins at the first character of *source\_string*. If *position* is greater than the number of characters in *source\_string*, the result is an empty string ("").

*occurrence*

A positive integer that indicates which occurrence of the pattern to use. REGEXP\_SUBSTR skips the first *occurrence* -1 matches. The default is 1. If *occurrence* is less than 1 or greater than the number of characters in *source\_string*, the search is ignored and the result is NULL.

*parameters*

One or more string literals that indicate how the function matches the pattern. The possible values are the following:

- c – Perform case-sensitive matching. The default is to use case-sensitive matching.
- i – Perform case-insensitive matching.
- e – Extract a substring using a subexpression.

If *pattern* includes a subexpression, REGEXP\_SUBSTR matches a substring using the first subexpression in *pattern*. A subexpression is an expression within the pattern that is bracketed with parentheses. For example, for the pattern 'This is a (\w+)' matches the first expression with the string 'This is a ' followed by a word. Instead of returning *pattern*, REGEXP\_SUBSTR with the e parameter returns only the string inside the subexpression.

REGEXP\_SUBSTR considers only the first subexpression; additional subexpressions are ignored. If the pattern doesn't have a subexpression, REGEXP\_SUBSTR ignores the 'e' parameter.

- p – Interpret the pattern with Perl Compatible Regular Expression (PCRE) dialect.

## Return type

VARCHAR

## Example

The following example returns the portion of an email address between the @ character and the domain extension.

```
SELECT email, regexp_substr(email,'@[^.]*')
```

```
ORDER BY userid LIMIT 4;

email | regexp_substr

Etiam.laoreet.libero@sodalesMaurisblandit.edu | @sodalesMaurisblandit
Suspendisse.tristique@nonnisiAenean.edu | @nonnisiAenean
amet.faucibus.ut@condimentumegetvolutpat.ca | @condimentumegetvolutpat
sed@lacusUtne.ca | @lacusUtne
```

The following example returns the portion of the input corresponding to the first occurrence of the string FOX, using case-insensitive matching.

```
SELECT regexp_substr('the fox', 'FOX', 1, 1, 'i');

regexp_substr

fox
```

The following example returns the first portion of the input that begins with lowercase letters. This is functionally identical to the same SELECT statement without the c parameter.

```
SELECT regexp_substr('THE SECRET CODE IS THE LOWERCASE PART OF 1931abc0EZ.', '[a-z]+', 1,
1, 'c');

regexp_substr

abc
```

The following example uses a pattern written in the PCRE dialect to locate words containing at least one number and one lowercase letter. It uses the ?= operator, which has a specific look-ahead connotation in PCRE. This example returns the portion of the input corresponding to the second such word.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^]*[a-z])(?=[^]*[0-9])[^]+', 1,
2, 'p');

regexp_substr

a1234
```

The following example uses a pattern written in the PCRE dialect to locate words containing at least one number and one lowercase letter. It uses the ?= operator, which has a specific look-ahead connotation in PCRE. This example returns the portion of the input corresponding to the second such word, but differs from the previous example in that it uses case-insensitive matching.

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^]*[a-z])(?=[^]*[0-9])[^]+', 1,
2, 'ip');

regexp_substr

A1234
```

The following example uses a subexpression to find the second string matching the pattern 'this is a (\w+)' using case-insensitive matching. It returns the subexpression inside the parentheses.

```
select regexp_substr(
 'This is a cat, this is a dog. This is a mouse.',
 'this is a (\w+)', 1, 2, 'ie');
```

```
regexp_substr

dog
```

## REPEAT function

Repeats a string the specified number of times. If the input parameter is numeric, REPEAT treats it as a string.

Synonym for [REPLICATE function \(p. 1310\)](#).

### Syntax

```
REPEAT(string, integer)
```

### Arguments

*string*

The first input parameter is the string to be repeated.

*integer*

The second parameter is an integer indicating the number of times to repeat the string.

### Return type

The REPEAT function returns a string.

### Examples

The following example repeats the value of the CATID column in the CATEGORY table three times:

```
select catid, repeat(catid,3)
from category
order by 1,2;

 catid | repeat
-----+-----
 1 | 111
 2 | 222
 3 | 333
 4 | 444
 5 | 555
 6 | 666
 7 | 777
 8 | 888
 9 | 999
 10 | 101010
 11 | 111111
(11 rows)
```

## REPLACE function

Replaces all occurrences of a set of characters within an existing string with other specified characters.

REPLACE is similar to the [TRANSLATE function \(p. 1321\)](#) and the [REGEXP\\_REPLACE function \(p. 1304\)](#), except that TRANSLATE makes multiple single-character substitutions and REGEXP\_REPLACE lets you search a string for a regular expression pattern, while REPLACE substitutes one entire string with another string.

## Syntax

```
REPLACE(string1, old_chars, new_chars)
```

## Arguments

*string*

CHAR or VARCHAR string to be searched

*old\_chars*

CHAR or VARCHAR string to replace.

*new\_chars*

New CHAR or VARCHAR string replacing the *old\_string*.

## Return type

VARCHAR

If either *old\_chars* or *new\_chars* is NULL, the return is NULL.

## Examples

The following example converts the string Shows to Theatre in the CATGROUP field:

```
select catid, catgroup,
 replace(catgroup, 'Shows', 'Theatre')
 from category
 order by 1,2,3;

 catid | catgroup | replace
-----+-----+-----
 1 | Sports | Sports
 2 | Sports | Sports
 3 | Sports | Sports
 4 | Sports | Sports
 5 | Sports | Sports
 6 | Shows | Theatre
 7 | Shows | Theatre
 8 | Shows | Theatre
 9 | Concerts | Concerts
 10 | Concerts | Concerts
 11 | Concerts | Concerts
(11 rows)
```

## REPLICATE function

Synonym for the REPEAT function.

See [REPEAT function \(p. 1309\)](#).

## REVERSE function

The REVERSE function operates on a string and returns the characters in reverse order. For example, `reverse('abcde')` returns edcba. This function works on numeric and date data types as well as character data types; however, in most cases it has practical value for character strings.

### Syntax

```
REVERSE (expression)
```

### Argument

*expression*

An expression with a character, date, timestamp, or numeric data type that represents the target of the character reversal. All expressions are implicitly converted to variable-length character strings. Trailing blanks in fixed-width character strings are ignored.

### Return type

REVERSE returns a VARCHAR.

### Examples

Select five distinct city names and their corresponding reversed names from the USERS table:

```
select distinct city as cityname, reverse(cityname)
from users order by city limit 5;

cityname | reverse
-----+-----
Aberdeen | needrebA
Abilene | enelibA
Ada | adA
Agat | tagA
Agawam | mawagA
(5 rows)
```

Select five sales IDs and their corresponding reversed IDs cast as character strings:

```
select salesid, reverse(salesid)::varchar
from sales order by salesid desc limit 5;

salesid | reverse
-----+-----
172456 | 654271
172455 | 554271
172454 | 454271
172453 | 354271
172452 | 254271
(5 rows)
```

## RTRIM function

The RTRIM function trims a specified set of characters from the end of a string. Removes the longest string containing only characters in the trim characters list. Trimming is complete when a trim character does not appear in the input string.

## Syntax

```
RTRIM(string, trim_chars)
```

## Arguments

### *string*

A string column, expression, or string literal to be trimmed.

### *trim\_chars*

A string column, expression, or string literal that represents the characters to be trimmed from the end of *string*. If not specified, a space is used as the trim character.

## Return type

A string that is the same data type as the *string* argument.

## Example

The following example trims leading and trailing blanks from the string ' abc ':

```
select ' abc ' as untrim, rtrim(' abc ') as trim;

untrim | trim
-----+-----
abc | abc
```

The following example removes the trailing 'xyz' strings from the string 'xyzaxyzbxyzcxyz'. The trailing occurrences of 'xyz' are removed, but occurrences that are internal within the string are not removed.

```
select 'xyzaxyzbxyzcxyz' as untrim,
rtrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

untrim | trim
-----+-----
xyzaxyzbxyzcxyz | xyzaxyzbxyzc
```

The following example removes the trailing parts from the string 'setuphistorycassettes' that match any of the characters in the *trim\_chars* list 'tes'. Any t, e, or s that occur before another character that is not in the *trim\_chars* list at the ending of the input string are removed.

```
SELECT rtrim('setuphistorycassettes', 'tes');

rtrim

setuphistoryca
```

The following example trims the characters 'Park' from the end of VENUENAME where present:

```
select venueid, venuename, rtrim(venuename, 'Park')
from venue
order by 1, 2, 3
```

```
limit 10;
```

| venueid | venuename                  | rtrim                   |
|---------|----------------------------|-------------------------|
| 1       | Toyota Park                | Toyota                  |
| 2       | Columbus Crew Stadium      | Columbus Crew Stadium   |
| 3       | RFK Stadium                | RFK Stadium             |
| 4       | CommunityAmerica Ballpark  | CommunityAmerica Ballp  |
| 5       | Gillette Stadium           | Gillette Stadium        |
| 6       | New York Giants Stadium    | New York Giants Stadium |
| 7       | BMO Field                  | BMO Field               |
| 8       | The Home Depot Center      | The Home Depot Cente    |
| 9       | Dick's Sporting Goods Park | Dick's Sporting Goods   |
| 10      | Pizza Hut Park             | Pizza Hut               |

Note that RTRIM removes any of the characters P, a, r, or k when they appear at the end of a VENUENAME.

## SOUNDEX function

The SOUNDEX function returns the American Soundex value consisting of the first letter followed by a 3-digit encoding of the sounds that represent the English pronunciation of the string that you specify.

### Syntax

```
SOUNDEX(string)
```

### Arguments

*string*

You specify a CHAR or VARCHAR string that you want to convert to an American Soundex code value.

### Return type

The SOUNDEX function returns a VARCHAR(4) string consisting of a capital letter followed by a three-digit encoding of the sounds that represent the English pronunciation.

### Usage notes

The SOUNDEX function converts only English alphabetical lowercase and uppercase ASCII characters, including a–z and A–Z. SOUNDEX ignores other characters. SOUNDEX returns a single Soundex value for a string of multiple words separated by spaces.

```
select soundex('AWS Amazon');
```

```
soundex

A252
```

SOUNDEX returns an empty string if the input string doesn't contain any English letters.

```
select soundex('+-*/%');
```

```
soundex

```

## Example

The following example returns the Soundex A525 for the word Amazon.

```
select soundex('Amazon');
```

```
soundex

A525
```

## SPLIT\_PART function

Splits a string on the specified delimiter and returns the part at the specified position.

### Syntax

```
SPLIT_PART(string, delimiter, position)
```

### Arguments

*string*

A string column, expression, or string literal to be split. The string can be CHAR or VARCHAR.

*delimiter*

The delimiter string indicating sections of the input *string*.

If *delimiter* is a literal, enclose it in single quotation marks.

*position*

Position of the portion of *string* to return (counting from 1). Must be an integer greater than 0.

If *position* is larger than the number of string portions, SPLIT\_PART returns an empty string. If *delimiter* is not found in *string*, then the returned value contains the contents of the specified part, which might be the entire *string* or an empty value.

### Return type

A CHAR or VARCHAR string, the same as the *string* parameter.

### Examples

The following example splits a string literal into parts using the \$ delimiter and returns the second part.

```
select split_part('abcdefghi','$',2)
split_part

def
```

The following example splits a string literal into parts using the \$ delimiter. It returns an empty string because part 4 is not found.

```
select split_part('abcdefghi','$',4)
split_part

```

The following example splits a string literal into parts using the # delimiter. It returns the entire string, which is the first part, because the delimiter is not found.

```
select split_part('abcdefghi','#',1)
split_part

abcdefghi
```

The following example splits the timestamp field LISTTIME into year, month, and day components.

```
select listtime, split_part(listtime,'-',1) as year,
 split_part(listtime,'-',2) as month,
 split_part(split_part(listtime,'-',3),' ',1) as day
 from listing limit 5;

listtime | year | month | day
-----+-----+-----+
2008-03-05 12:25:29 | 2008 | 03 | 05
2008-09-09 08:03:36 | 2008 | 09 | 09
2008-09-26 05:43:12 | 2008 | 09 | 26
2008-10-04 02:00:30 | 2008 | 10 | 04
2008-01-06 08:33:11 | 2008 | 01 | 06
```

The following example selects the LISTTIME timestamp field and splits it on the '-' character to get the month (the second part of the LISTTIME string), then counts the number of entries for each month:

```
select split_part(listtime,'-',2) as month, count(*)
 from listing
 group by split_part(listtime,'-',2)
 order by 1, 2;

month | count
-----+-----
 01 | 18543
 02 | 16620
 03 | 17594
 04 | 16822
 05 | 17618
 06 | 17158
 07 | 17626
 08 | 17881
 09 | 17378
 10 | 17756
 11 | 12912
 12 | 4589
```

## STRPOS function

Returns the position of a substring within a specified string.

Synonym of [CHARINDEX function \(p. 1283\)](#) and [POSITION function \(p. 1298\)](#).

## Syntax

```
STRPOS(string, substring)
```

## Arguments

*string*

The first input parameter is the string to be searched.

*substring*

The second parameter is the substring to search for within the *string*.

## Return type

The STRPOS function returns an integer corresponding to the position of the substring (one-based, not zero-based). The position is based on the number of characters, not bytes, so that multi-byte characters are counted as single characters.

## Usage notes

STRPOS returns 0 if the *substring* is not found within the *string*:

```
select strpos('dogfish', 'fist');
strpos

0
(1 row)
```

## Examples

The following example shows the position of the string fish within the word dogfish:

```
select strpos('dogfish', 'fish');
strpos

4
(1 row)
```

The following example returns the number of sales transactions with a COMMISSION over 999.00 from the SALES table:

```
select distinct strpos(commission, '.'),
count(strpos(commission, '.'))
from sales
where strpos(commission, '.') > 4
group by strpos(commission, '.')
order by 1, 2;

strpos | count
-----+-----
5 | 629
(1 row)
```

## STRTOL function

Converts a string expression of a number of the specified base to the equivalent integer value. The converted value must be within the signed 64-bit range.

### Syntax

```
STRTOL(num_string, base)
```

### Arguments

*num\_string*

String expression of a number to be converted. If *num\_string* is empty ( '') or begins with the null character ('\0'), the converted value is 0. If *num\_string* is a column containing a NULL value, STRTOL returns NULL. The string can begin with any amount of white space, optionally followed by a single plus '+' or minus '-' sign to indicate positive or negative. The default is '+'. If *base* is 16, the string can optionally begin with '0x'.

*base*

Integer between 2 and 36.

### Return type

BIGINT. If *num\_string* is null, returns NULL.

### Examples

The following examples convert string and base value pairs to integers:

```
select strtol('0xf',16);
 strtol

 15
(1 row)

select strtol('abcd1234',16);
 strtol

 2882343476
(1 row)

select strtol('1234567', 10);
 strtol

 1234567
(1 row)

select strtol('1234567', 8);
 strtol

 342391
(1 row)

select strtol('110101', 2);
 strtol

 53
```

```
select strtol('\0', 2);
 strtol

 0
```

## SUBSTR function

Synonym of the SUBSTRING function.

See [SUBSTRING function \(p. 1318\)](#).

## SUBSTRING function

Returns the subset of a string based on the specified start position.

If the input is a character string, the start position and number of characters extracted are based on characters, not bytes, so that multi-byte characters are counted as single characters. If the input is a binary expression, the start position and extracted substring are based on bytes. You can't specify a negative length, but you can specify a negative starting position.

### Syntax

```
SUBSTRING(character_string FROM start_position [FOR number_characters])
```

```
SUBSTRING(character_string, start_position, number_characters)
```

```
SUBSTRING(binary_expression, start_byte, number_bytes)
```

```
SUBSTRING(binary_expression, start_byte)
```

### Arguments

*character\_string*

The string to be searched. Non-character data types are treated like a string.

*start\_position*

The position within the string to begin the extraction, starting at 1. The *start\_position* is based on the number of characters, not bytes, so that multi-byte characters are counted as single characters.

This number can be negative.

*number\_characters*

The number of characters to extract (the length of the substring). The *number\_characters* is based on the number of characters, not bytes, so that multi-byte characters are counted as single characters.

This number cannot be negative.

*binary\_expression*

The *binary\_expression* of data type VARBYTE to be searched.

*start\_byte*

The position within the *binary\_expression* to begin the extraction, starting at 1. This number can be negative.

*number\_bytes*

The number of bytes to extract, that is, the length of the substring. This number can't be negative.

## Return type

VARCHAR or VARBYTE depending on the input.

## Usage notes for character strings

The following example returns a four-character string beginning with the sixth character.

```
select substring('caterpillar',6,4);
substring

pill
(1 row)
```

If the `start_position + number_characters` exceeds the length of the `string`, SUBSTRING returns a substring starting from the `start_position` until the end of the string. For example:

```
select substring('caterpillar',6,8);
substring

pillar
(1 row)
```

If the `start_position` is negative or 0, the SUBSTRING function returns a substring beginning at the first character of string with a length of `start_position + number_characters -1`. For example:

```
select substring('caterpillar',-2,6);
substring

cat
(1 row)
```

If `start_position + number_characters -1` is less than or equal to zero, SUBSTRING returns an empty string. For example:

```
select substring('caterpillar',-5,4);
substring

(1 row)
```

## Examples

The following example returns the month from the LISTTIME string in the LISTING table:

```
select listid, listtime,
substring(listtime, 6, 2) as month
from listing
order by 1, 2, 3
limit 10;

listid | listtime | month
-----+-----+-----+
 1 | 2008-01-24 06:43:29 | 01
 2 | 2008-03-05 12:25:29 | 03
 3 | 2008-11-01 07:35:33 | 11
 4 | 2008-05-24 01:18:37 | 05
 5 | 2008-05-17 02:29:11 | 05
 6 | 2008-08-15 02:08:13 | 08
```

```

7 | 2008-11-15 09:38:15 | 11
8 | 2008-11-09 05:07:30 | 11
9 | 2008-09-09 08:03:36 | 09
10 | 2008-06-17 09:44:54 | 06
(10 rows)

```

The following example is the same as above, but uses the FROM...FOR option:

```

select listid, listtime,
substring(listtime from 6 for 2) as month
from listing
order by 1, 2, 3
limit 10;

listid | listtime | month
-----+-----+-----+
1 | 2008-01-24 06:43:29 | 01
2 | 2008-03-05 12:25:29 | 03
3 | 2008-11-01 07:35:33 | 11
4 | 2008-05-24 01:18:37 | 05
5 | 2008-05-17 02:29:11 | 05
6 | 2008-08-15 02:08:13 | 08
7 | 2008-11-15 09:38:15 | 11
8 | 2008-11-09 05:07:30 | 11
9 | 2008-09-09 08:03:36 | 09
10 | 2008-06-17 09:44:54 | 06
(10 rows)

```

You can't use SUBSTRING to predictably extract the prefix of a string that might contain multi-byte characters because you need to specify the length of a multi-byte string based on the number of bytes, not the number of characters. To extract the beginning segment of a string based on the length in bytes, you can CAST the string as VARCHAR(*byte\_length*) to truncate the string, where *byte\_length* is the required length. The following example extracts the first 5 bytes from the string 'Fourscore and seven'.

```

select cast('Fourscore and seven' as varchar(5));
varchar

Fours

```

The following example shows a negative start position of a binary value abc. Because the start position is -3, the substring is extracted from the beginning of the binary value. The result is automatically shown as the hexadecimal representation of the binary substring.

```

select substring('abc'::varbyte, -3);
substring

616263

```

The following example shows a 1 for the start position of a binary value abc. Because because there is no length specified, the string is extracted from the start position to the end of the string. The result is automatically shown as the hexadecimal representation of the binary substring.

```

select substring('abc'::varbyte, 1);
substring

```

616263

The following example shows a 3 for the start position of a binary value abc. Because because there is no length specified, the string is extracted from the start position to the end of the string. The result is automatically shown as the hexadecimal representation of the binary substring.

```
select substring('abc'::varbyte, 3);

substring

63
```

The following example shows a 2 for the start position of a binary value abc. The string is extracted from the start position to position 10, but the end of the string is at position 3. The result is automatically shown as the hexadecimal representation of the binary substring.

```
select substring('abc'::varbyte, 2, 10);

substring

6263
```

The following example shows a 2 for the start position of a binary value abc. The string is extracted from the start position for 1 byte. The result is automatically shown as the hexadecimal representation of the binary substring.

```
select substring('abc'::varbyte, 2, 1);

substring

62
```

## TEXTLEN function

Synonym of LEN function.

See [LEN function \(p. 1292\)](#).

## TRANSLATE function

For a given expression, replaces all occurrences of specified characters with specified substitutes. Existing characters are mapped to replacement characters by their positions in the *characters\_to\_replace* and *characters\_to\_substitute* arguments. If more characters are specified in the *characters\_to\_replace* argument than in the *characters\_to\_substitute* argument, the extra characters from the *characters\_to\_replace* argument are omitted in the return value.

TRANSLATE is similar to the [REPLACE function \(p. 1309\)](#) and the [REGEXP\\_REPLACE function \(p. 1304\)](#), except that REPLACE substitutes one entire string with another string and REGEXP\_REPLACE lets you search a string for a regular expression pattern, while TRANSLATE makes multiple single-character substitutions.

If any argument is null, the return is NULL.

### Syntax

```
TRANSLATE (expression, characters_to_replace, characters_to_substitute)
```

## Arguments

*expression*

The expression to be translated.

*characters\_to\_replace*

A string containing the characters to be replaced.

*characters\_to\_substitute*

A string containing the characters to substitute.

## Return type

VARCHAR

## Examples

The following example replaces several characters in a string:

```
select translate('mint tea', 'inea', 'osin');
translate

most tin
```

The following example replaces the at sign (@) with a period for all values in a column:

```
select email, translate(email, '@', '.') as obfuscated_email
from users limit 10;

email obfuscated_email

Etiam.laoreet.libero@sodalesMaurisblandit.edu
Etiam.laoreet.libero.sodalesMaurisblandit.edu
amet.faucibus.ut@condimentumegetvolutpat.ca amet.faucibus.ut.condimentumegetvolutpat.ca
turpis@accumsanlaoreet.org turpis.accumsanlaoreet.org
ullamcorper.nisl@Cras.edu ullamcorper.nisl.Cras.edu
arcu.Curabitur@senectusetnetus.com arcu.Curabitur.senectusetnetus.com
ac@velit.ca ac.velit.ca
Aliquam.vulputate.ullamcorper@amalesuada.org
Aliquam.vulputate.ullamcorper.amalesuada.org
vel.est@elitegestas.edu vel.est.elitegestas.edu
dolor.nonummy@ipsumdolorsit.ca dolor.nonummy.ipsumdolorsit.ca
et@Nunclaoreet.ca et.Nunclaoreet.ca
```

The following example replaces spaces with underscores and strips out periods for all values in a column:

```
select city, translate(city, ' .', '_') from users
where city like 'Sain%' or city like 'St%'
group by city
order by city;

city translate

Saint Albans Saint_Albans
Saint Cloud Saint_Cloud
Saint Joseph Saint_Joseph
Saint Louis Saint_Louis
```

|                |               |
|----------------|---------------|
| Saint Paul     | Saint_Paul    |
| St. George     | St_George     |
| St. Marys      | St_Marys      |
| St. Petersburg | St_Petersburg |
| Stafford       | Stafford      |
| Stamford       | Stamford      |
| Stanton        | Stanton       |
| Starkville     | Starkville    |
| Statesboro     | Statesboro    |
| Staunton       | Staunton      |
| Steubenville   | Steubenville  |
| Stevens Point  | Stevens_Point |
| Stillwater     | Stillwater    |
| Stockton       | Stockton      |
| Sturgis        | Sturgis       |

## TRIM function

Trims a string by removing leading and trailing blanks or by removing leading and trailing characters that match an optional specified string.

### Syntax

```
TRIM([BOTH] [trim_chars FROM] string)
```

### Arguments

*trim\_chars*

(Optional) The characters to be trimmed from the string. If this parameter is omitted, blanks are trimmed.

*string*

The string to be trimmed.

### Return type

The TRIM function returns a VARCHAR or CHAR string. If you use the TRIM function with a SQL command, Amazon Redshift implicitly converts the results to VARCHAR. If you use the TRIM function in the SELECT list for a SQL function, Amazon Redshift does not implicitly convert the results, and you might need to perform an explicit conversion to avoid a data type mismatch error. See the [CAST function \(p. 1066\)](#) and [CONVERT function \(p. 1069\)](#) functions for information about explicit conversions.

### Example

The following example trims leading and trailing blanks from the string ' abc ':

```
select ' abc ' as untrim, trim(' abc ') as trim;

untrim | trim
-----+-----
abc | abc
```

The following example removes the double quotation marks that surround the string "dog":

```
select trim('"' FROM '"dog"');
```

```
btrim

dog
```

TRIM removes any of the characters in *trim\_chars* when they appear at the beginning of *string*. The following example trims the characters 'C', 'D', and 'G' when they appear at the beginning of VENUENAME, which is a VARCHAR column.

| venueid | venuename                  | btrim                     |
|---------|----------------------------|---------------------------|
| 121     | ATT Park                   | ATT Park                  |
| 109     | Citizens Bank Park         | itizens Bank Park         |
| 102     | Comerica Park              | omerica Park              |
| 9       | Dick's Sporting Goods Park | ick's Sporting Goods Park |
| 97      | Fenway Park                | Fenway Park               |
| 112     | Great American Ball Park   | reat American Ball Park   |
| 114     | Miller Park                | Miller Park               |

## UPPER function

Converts a string to uppercase. UPPER supports UTF-8 multibyte characters, up to a maximum of four bytes per character.

### Syntax

```
UPPER(string)
```

### Arguments

*string*

The input parameter is a VARCHAR string (or any other data type, such as CHAR, that can be implicitly converted to VARCHAR).

### Return type

The UPPER function returns a character string that is the same data type as the input string.

### Examples

The following example converts the CATNAME field to uppercase:

| catname   | upper     |
|-----------|-----------|
| Classical | CLASSICAL |
| Jazz      | JAZZ      |
| MLB       | MLB       |
| MLS       | MLS       |
| Musicals  | MUSICALS  |

|           |       |
|-----------|-------|
| NBA       | NBA   |
| NFL       | NFL   |
| NHL       | NHL   |
| Opera     | OPERA |
| Plays     | PLAYS |
| Pop       | POP   |
| (11 rows) |       |

## SUPER type information functions

Following, you can find a description for the type information functions for SQL that Amazon Redshift supports to derive the dynamic information from inputs of the SUPER data type.

### Topics

- [DECIMAL\\_PRECISION function \(p. 1325\)](#)
- [DECIMAL\\_SCALE function \(p. 1326\)](#)
- [IS\\_ARRAY function \(p. 1326\)](#)
- [IS\\_BIGINT function \(p. 1327\)](#)
- [IS\\_BOOLEAN function \(p. 1328\)](#)
- [IS\\_CHAR function \(p. 1328\)](#)
- [IS\\_DECIMAL function \(p. 1329\)](#)
- [IS\\_FLOAT function \(p. 1329\)](#)
- [IS\\_INTEGER function \(p. 1330\)](#)
- [IS\\_OBJECT function \(p. 1331\)](#)
- [IS\\_SCALAR function \(p. 1331\)](#)
- [IS\\_SMALLINT function \(p. 1332\)](#)
- [IS\\_VARCHAR function \(p. 1333\)](#)
- [JSON\\_SIZE function \(p. 1333\)](#)
- [JSON\\_TYPEOF function \(p. 1334\)](#)
- [SIZE \(p. 1335\)](#)

## DECIMAL\_PRECISION function

Checks the precision of the maximum total number of decimal digits to be stored. This number includes both the left and right digits of the decimal point. The range of the precision is from 1 to 38, with a default of 38.

### Syntax

```
DECIMAL_PRECISION (super_expression)
```

### Arguments

*super\_expression*

A SUPER expression or column.

### Returns

Integer

## Example

The following query applies the DECIMAL\_PRECISION function to the table t.

```
CREATE TABLE t(s super);
INSERT INTO t VALUES (3.14159);
SELECT decimal_precision(s) FROM t;
decimal_precision

6
(1 row)
```

## DECIMAL\_SCALE function

Checks the number of decimal digits to be stored to the right of the decimal point. The range of the scale is from 0 to the precision point, with a default of 0.

### Syntax

```
DECIMAL_SCALE (super_expression)
```

### Arguments

*super\_expression*

A SUPER expression or column.

### Returns

Integer

### Example

The following query applies the DECIMAL\_SCALE function to the table t.

```
CREATE TABLE t(s super);
INSERT INTO t VALUES (3.14159);
SELECT decimal_scale(s) FROM t;
decimal_scale

5
(1 row)
```

## IS\_ARRAY function

Checks whether a variable is an array. The function returns true if the variable is an array. The function also includes empty arrays. Otherwise, the function returns false for all other values, including null.

### Syntax

```
IS_ARRAY (super_expression)
```

## Arguments

*super\_expression*

A SUPER expression or column.

## Returns

Boolean

## Example

The following query shows an IS\_ARRAY function.

```
SELECT is_array(JSON_PARSE('[1,2]'));
is_array

 True
(1 row)
```

## IS\_BIGINT function

Checks whether a value is a BIGINT. The IS\_BIGINT function returns true for numbers of scale 0 in the 64-bit range. Otherwise, the function returns false for all other values, including null and floating point numbers.

The IS\_BIGINT function is a superset of IS\_INTEGER.

## Syntax

```
IS_BIGINT (super_expression)
```

## Arguments

*super\_expression*

A SUPER expression or column.

## Returns

Boolean

## Example

The following query shows an IS\_BIGINT function.

```
CREATE TABLE t(s super);
INSERT INTO t VALUES (5);
SELECT s, is_bigint(s) FROM t;
s | is_bigint
---+-----
 5 | True
(1 row)
```

## IS\_BOOLEAN function

Checks whether a value is a BOOLEAN. The IS\_BOOLEAN function returns true for constant JSON Booleans. The function returns false for any other values, including null.

### Syntax

```
IS_BOOLEAN (super_expression)
```

### Arguments

*super\_expression*

A SUPER expression or column.

### Returns

Boolean

### Example

The following query shows an IS\_BOOLEAN function.

```
CREATE TABLE t(s super);

INSERT INTO t VALUES (TRUE);

SELECT s, is_boolean(s) FROM t;
 s | is_boolean
-----+-----
"true" | True
(1 row)
```

## IS\_CHAR function

Checks whether a value is a CHAR. The IS\_CHAR function returns true for strings that have only ASCII characters, because the CHAR type can store only characters that are in the ASCII format. The function returns false for any other values.

### Syntax

```
IS_CHAR (super_expression)
```

### Arguments

*super\_expression*

A SUPER expression or column.

### Returns

Boolean

## Example

The following query shows an IS\_CHAR function.

```
CREATE TABLE t(s super);
INSERT INTO t VALUES ('t');
SELECT s, is_char(s) FROM t;
s | is_char
----+-----
"t" | True
(1 row);
```

## IS\_DECIMAL function

Checks whether a value is a DECIMAL. The IS\_DECIMAL function returns true for numbers that are not floating points. The function returns false for any other values, including null.

The IS\_DECIMAL function is a superset of IS\_BIGINT.

### Syntax

```
IS_DECIMAL (super_expression)
```

### Arguments

*super\_expression*

A SUPER expression or column.

### Returns

Boolean

## Example

The following query shows an IS\_DECIMAL function.

```
CREATE TABLE t(s super);
INSERT INTO t VALUES (1.22);
SELECT s, is_decimal(s) FROM t;
s | is_decimal
----+-----
1.22 | True
(1 row);
```

## IS\_FLOAT function

Checks whether a value is a floating point number. The IS\_FLOAT function returns true for floating point numbers (FLOAT4 and FLOAT8). The function returns false for any other values.

The set of IS\_DECIMAL the set of IS\_FLOAT are disjoint.

## Syntax

```
IS_FLOAT (super_expression)
```

## Arguments

*super\_expression*

A SUPER expression or column.

## Returns

Boolean

## Example

The following query shows an IS\_FLOAT function.

```
CREATE TABLE t(s super);

INSERT INTO t VALUES(2.22::FLOAT);

SELECT s, is_float(s) FROM t;

s | is_float
-----+-----
2.22e+0 | True
(1 row)
```

## IS\_INTEGER function

Returns true for numbers of scale 0 in the 32-bit range, and false for anything else (including null and floating point numbers).

The IS\_INTEGER function is a superset of the IS\_SMALLINT function.

## Syntax

```
IS_INTEGER (super_expression)
```

## Arguments

*super\_expression*

A SUPER expression or column.

## Returns

Boolean

## Example

The following example shows an IS\_INTEGER function.

```
CREATE TABLE t(s super);

INSERT INTO t VALUES (5);

SELECT s, is_integer(s) FROM t;

s | is_integer
---+-----
5 | True
(1 row)
```

## IS\_OBJECT function

Checks whether a variable is an object. The IS\_OBJECT function returns true for objects, including empty objects. The function returns false for any other values, including null.

### Syntax

|                                                  |
|--------------------------------------------------|
| <code>IS_OBJECT (<i>super_expression</i>)</code> |
|--------------------------------------------------|

### Arguments

*super\_expression*

A SUPER expression or column.

### Returns

Boolean

### Example

The following query shows an IS\_OBJECT function.

```
CREATE TABLE t(s super);

INSERT INTO t VALUES (JSON_PARSE('{"name": "Joe"}'));

SELECT s, is_object(s) FROM t;

s | is_object
-----+-----
{"name":"Joe"} | True
(1 row)
```

## IS\_SCALAR function

Checks whether a variable is a scalar. The IS\_SCALAR function returns true for any value that is not an array or an object. The function returns false for any other values, including null.

The set of IS\_ARRAY, IS\_OBJECT, and IS\_SCALAR cover all values except nulls.

### Syntax

|                                                  |
|--------------------------------------------------|
| <code>IS_SCALAR (<i>super_expression</i>)</code> |
|--------------------------------------------------|

## Arguments

*super\_expression*

A SUPER expression or column.

## Returns

Boolean

## Example

The following query shows an IS\_SCALAR function.

```
CREATE TABLE t(s super);
INSERT INTO t VALUES (JSON_PARSE('{"name": "Joe"}'));
SELECT s, is_scalar(s.name) FROM t;
s | is_scalar
-----+-----
{"name":"Joe"} | True
(1 row)
```

## IS\_SMALLINT function

Checks whether a variable is a SMALLINT. The IS\_SMALLINT function returns true for numbers of scale 0 in the 16-bit range. The function returns false for any other values, including null and floating point numbers.

## Syntax

```
IS_SMALLINT (super_expression)
```

## Arguments

*super\_expression*

A SUPER expression or column.

## Returns

Boolean

## Example

The following query shows an IS\_SMALLINT function.

```
CREATE TABLE t(s super);
INSERT INTO t VALUES (5);
SELECT s, is_smallint(s) FROM t;
```

```

s | is_smallint
---+-----
5 | True
(1 row)

```

## IS\_VARCHAR function

Checks whether a variable is a VARCHAR. The IS\_VARCHAR function returns true for all strings. The function returns false for any other values.

The IS\_VARCHAR function is a superset of the IS\_CHAR function.

### Syntax

```
IS_VARCHAR (super_expression)
```

### Arguments

*super\_expression*

A SUPER expression or column.

### Returns

Boolean

### Example

The following query shows an IS\_VARCHAR function.

```

CREATE TABLE t(s super);

INSERT INTO t VALUES ('abc');

SELECT s, is_varchar(s) FROM t;

s | is_varchar
---+-----
"abc" | True
(1 row)

```

## JSON\_SIZE function

The JSON\_SIZE function returns the number of bytes in the given SUPER expression when serialized into a string.

### Syntax

```
JSON_SIZE(super_expression)
```

### Arguments

*super\_expression*

A SUPER constant or expression.

## Returns

The JSON\_SIZE function returns an integer indicating the number of bytes in the input string. This value is different from the number of characters. For example, the UTF-8 character #, a black dot, is 3 bytes in size.

## Usage Notes

JSON\_SIZE(x) is functionally identical to OCTET\_LENGTH(JSON\_SERIALIZE). However, note that JSON\_SERIALIZE returns an error when the provided SUPER expression would exceed the VARCHAR limit of the system when serialized. JSON\_SIZE does not have this limitation.

## Examples

The following example returns the length of a SUPER value serialized to a string.

```
SELECT JSON_SIZE(JSON_PARSE('[10001,10002,"#"]'));
 json_size

 19
(1 row)
```

The provided SUPER expression is 17 characters long, but # is a 3-byte character, so JSON\_SIZE returns 19.

## JSON\_TYPEOF function

The JSON\_TYPEOF scalar function returns a VARCHAR with values boolean, number, string, object, array, or null, depending on the dynamic type of the SUPER value.

## Syntax

```
JSON_TYPEOF (super_expression)
```

## Arguments

*super\_expression*

A SUPER expression or column.

## Returns

Varchar

## Examples

The following queries show a JSON\_TYPEOF function.

```
SELECT JSON_TYPEOF(ARRAY(1,2));
 json_typeof

 array
(1 row)
```

```
SELECT JSON_TYPEOF(JSON_PARSE('{"name":"Joe"}'));
```

```
json_typeof

object
(1 row)
```

## SIZE

Returns the binary in-memory size of a SUPER type constant or expression as an integer.

### Syntax

```
SIZE(super_expression)
```

### Parameters

*super\_expression*

A SUPER type constant or expression.

### Returns

Integer

### Example

The following example uses SIZE to get the in-memory size of several SUPER type expressions.

First, create and populate a table.

```
CREATE TABLE test_super_size(a super);

INSERT INTO test_super_size
VALUES
(
 (null),
 (TRUE),
 (json_parse('[0,1,2,3]')),
 (json_parse('{"a":0,"b":1,"c":2,"d":3}'))
);
```

Running SIZE returns the binary size of SUPER type expressions as an integer.

```
SELECT a, SIZE(a)
FROM test_super_size
ORDER BY 2, 1;

 a | size
-----+-----
true | 4
[0,1,2,3] | 23
{"a":0,"b":1,"c":2,"d":3} | 52
(4 rows)
```

## Window functions

By using window functions, you can create analytic business queries more efficiently. Window functions operate on a partition or "window" of a result set, and return a value for every row in that window. In

contrast, non-windowed functions perform their calculations with respect to every row in the result set. Unlike group functions that aggregate result rows, window functions retain all rows in the table expression.

The values returned are calculated by using values from the sets of rows in that window. For each row in the table, the window defines a set of rows that is used to compute additional attributes. A window is defined using a window specification (the OVER clause), and is based on three main concepts:

- *Window partitioning*, which forms groups of rows (PARTITION clause)
- *Window ordering*, which defines an order or sequence of rows within each partition (ORDER BY clause)
- *Window frames*, which are defined relative to each row to further restrict the set of rows (ROWS specification)

Window functions are the last set of operations performed in a query except for the final ORDER BY clause. All joins and all WHERE, GROUP BY, and HAVING clauses are completed before the window functions are processed. Therefore, window functions can appear only in the select list or ORDER BY clause. You can use multiple window functions within a single query with different frame clauses. You can also use window functions in other scalar expressions, such as CASE.

## Window function syntax summary

Window functions follow a standard syntax, which is as follows.

```
function (expression) OVER (
[PARTITION BY expr_list]
[ORDER BY order_list [frame_clause]])
```

Here, *function* is one of the functions described in this section.

The *expr\_list* is as follows.

```
expression | column_name [, expr_list]
```

The *order\_list* is as follows.

```
expression | column_name [ASC | DESC]
[NULLS FIRST | NULLS LAST]
[, order_list]
```

The *frame\_clause* is as follows.

```
ROWS
{ UNBOUNDED PRECEDING | unsigned_value PRECEDING | CURRENT ROW } |
{ BETWEEN
{ UNBOUNDED PRECEDING | unsigned_value { PRECEDING | FOLLOWING } | CURRENT ROW}
AND
{ UNBOUNDED FOLLOWING | unsigned_value { PRECEDING | FOLLOWING } | CURRENT ROW } }
```

## Arguments

*function*

The window function. For details, see the individual function descriptions.

## OVER

The clause that defines the window specification. The OVER clause is mandatory for window functions, and differentiates window functions from other SQL functions.

### PARTITION BY *expr\_list*

(Optional) The PARTITION BY clause subdivides the result set into partitions, much like the GROUP BY clause. If a partition clause is present, the function is calculated for the rows in each partition. If no partition clause is specified, a single partition contains the entire table, and the function is computed for that complete table.

The ranking functions DENSE\_RANK, NTILE, RANK, and ROW\_NUMBER require a global comparison of all the rows in the result set. When a PARTITION BY clause is used, the query optimizer can run each aggregation in parallel by spreading the workload across multiple slices according to the partitions. If the PARTITION BY clause is not present, the aggregation step must be run serially on a single slice, which can have a significant negative impact on performance, especially for large clusters.

Amazon Redshift doesn't support string literals in PARTITION BY clauses.

### ORDER BY *order\_list*

(Optional) The window function is applied to the rows within each partition sorted according to the order specification in ORDER BY. This ORDER BY clause is distinct from and completely unrelated to ORDER BY clauses in the *frame\_clause*. The ORDER BY clause can be used without the PARTITION BY clause.

For ranking functions, the ORDER BY clause identifies the measures for the ranking values. For aggregation functions, the partitioned rows must be ordered before the aggregate function is computed for each frame. For more about window function types, see [Window functions \(p. 1335\)](#).

Column identifiers or expressions that evaluate to column identifiers are required in the order list. Neither constants nor constant expressions can be used as substitutes for column names.

NULLS values are treated as their own group, sorted and ranked according to the NULLS FIRST or NULLS LAST option. By default, NULL values are sorted and ranked last in ASC ordering, and sorted and ranked first in DESC ordering.

Amazon Redshift doesn't support string literals in ORDER BY clauses.

If the ORDER BY clause is omitted, the order of the rows is nondeterministic.

#### Note

In any parallel system such as Amazon Redshift, when an ORDER BY clause doesn't produce a unique and total ordering of the data, the order of the rows is nondeterministic. That is, if the ORDER BY expression produces duplicate values (a partial ordering), the return order of those rows might vary from one run of Amazon Redshift to the next. In turn, window functions might return unexpected or inconsistent results. For more information, see [Unique ordering of data for window functions \(p. 1339\)](#).

### *column\_name*

Name of a column to be partitioned by or ordered by.

### ASC | DESC

Option that defines the sort order for the expression, as follows:

- ASC: ascending (for example, low to high for numeric values and 'A' to 'Z' for character strings). If no option is specified, data is sorted in ascending order by default.
- DESC: descending (high to low for numeric values; 'Z' to 'A' for strings).

## NULLS FIRST | NULLS LAST

Option that specifies whether NULLS should be ordered first, before non-null values, or last, after non-null values. By default, NULLS are sorted and ranked last in ASC ordering, and sorted and ranked first in DESC ordering.

### *frame\_clause*

For aggregate functions, the frame clause further refines the set of rows in a function's window when using ORDER BY. It enables you to include or exclude sets of rows within the ordered result. The frame clause consists of the ROWS keyword and associated specifiers.

The frame clause doesn't apply to ranking functions. Also, the frame clause isn't required when no ORDER BY clause is used in the OVER clause for an aggregate function. If an ORDER BY clause is used for an aggregate function, an explicit frame clause is required.

When no ORDER BY clause is specified, the implied frame is unbounded, equivalent to ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

### ROWS

This clause defines the window frame by specifying a physical offset from the current row.

This clause specifies the rows in the current window or partition that the value in the current row is to be combined with. It uses arguments that specify row position, which can be before or after the current row. The reference point for all window frames is the current row. Each row becomes the current row in turn as the window frame slides forward in the partition.

The frame can be a simple set of rows up to and including the current row.

```
{UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW}
```

Or it can be a set of rows between two boundaries.

```
BETWEEN
{ UNBOUNDED PRECEDING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
AND
{ UNBOUNDED FOLLOWING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
```

UNBOUNDED PRECEDING indicates that the window starts at the first row of the partition; *offset* PRECEDING indicates that the window starts a number of rows equivalent to the value of *offset* before the current row. UNBOUNDED PRECEDING is the default.

CURRENT ROW indicates the window begins or ends at the current row.

UNBOUNDED FOLLOWING indicates that the window ends at the last row of the partition; *offset* FOLLOWING indicates that the window ends a number of rows equivalent to the value of *offset* after the current row.

*offset* identifies a physical number of rows before or after the current row. In this case, *offset* must be a constant that evaluates to a positive numeric value. For example, 5 FOLLOWING ends the frame five rows after the current row.

Where BETWEEN is not specified, the frame is implicitly bounded by the current row. For example, ROWS 5 PRECEDING is equal to ROWS BETWEEN 5 PRECEDING AND CURRENT ROW. Also, ROWS UNBOUNDED FOLLOWING is equal to ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING.

#### Note

You can't specify a frame in which the starting boundary is greater than the ending boundary. For example, you can't specify any of the following frames.

```
between 5 following and 5 preceding
between current row and 2 preceding
between 3 following and current row
```

## Unique ordering of data for window functions

If an ORDER BY clause for a window function doesn't produce a unique and total ordering of the data, the order of the rows is nondeterministic. If the ORDER BY expression produces duplicate values (a partial ordering), the return order of those rows can vary in multiple runs. In this case, window functions can also return unexpected or inconsistent results.

For example, the following query returns different results over multiple runs. These different results occur because `order by dateid` doesn't produce a unique ordering of the data for the SUM window function.

```
select dateid, pricepaid,
sum(pricepaid) over(order by dateid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid | sumpaid
-----+-----+-----
1827 | 1730.00 | 1730.00
1827 | 708.00 | 2438.00
1827 | 234.00 | 2672.00
...
select dateid, pricepaid,
sum(pricepaid) over(order by dateid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid | sumpaid
-----+-----+-----
1827 | 234.00 | 234.00
1827 | 472.00 | 706.00
1827 | 347.00 | 1053.00
...
```

In this case, adding a second ORDER BY column to the window function can solve the problem.

```
select dateid, pricepaid,
sum(pricepaid) over(order by dateid, pricepaid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid | sumpaid
-----+-----+-----
1827 | 234.00 | 234.00
1827 | 337.00 | 571.00
1827 | 347.00 | 918.00
...
```

## Supported functions

Amazon Redshift supports two types of window functions: aggregate and ranking.

Following are the supported aggregate functions:

- [AVG window function \(p. 1341\)](#)
- [COUNT window function \(p. 1342\)](#)
- [CUME\\_DIST window function \(p. 1344\)](#)
- [DENSE\\_RANK window function \(p. 1345\)](#)
- [FIRST\\_VALUE window function \(p. 1347\)](#)
- [LAG window function \(p. 1349\)](#)
- [LAST\\_VALUE window function \(p. 1351\)](#)
- [LEAD window function \(p. 1352\)](#)
- [LISTAGG window function \(p. 1354\)](#)
- [MAX window function \(p. 1356\)](#)
- [MEDIAN window function \(p. 1358\)](#)
- [MIN window function \(p. 1360\)](#)
- [NTH\\_VALUE window function \(p. 1361\)](#)
- [PERCENTILE\\_CONT window function \(p. 1365\)](#)
- [PERCENTILE\\_DISC window function \(p. 1368\)](#)
- [RATIO\\_TO\\_REPORT window function \(p. 1371\)](#)
- [STDDEV\\_SAMP and STDDEV\\_POP window functions \(p. 1374\)](#) (STDDEV\_SAMP and STDDEV are synonyms)
- [SUM window function \(p. 1375\)](#)
- [VAR\\_SAMP and VAR\\_POP window functions \(p. 1378\)](#) (VAR\_SAMP and VARIANCE are synonyms)

Following are the supported ranking functions:

- [DENSE\\_RANK window function \(p. 1345\)](#)
- [NTILE window function \(p. 1363\)](#)
- [PERCENT\\_RANK window function \(p. 1364\)](#)
- [RANK window function \(p. 1369\)](#)
- [ROW\\_NUMBER window function \(p. 1373\)](#)

## Sample table for window function examples

You can find specific window function examples with each function description. Some of the examples use a table named WINSALES, which contains 11 rows, as shown following.

| SALESID | DATEID     | SELLERID | BUYERID | QTY | QTY_SHIPPED |
|---------|------------|----------|---------|-----|-------------|
| 30001   | 8/2/2003   | 3        | B       | 10  | 10          |
| 10001   | 12/24/2003 | 1        | C       | 10  | 10          |
| 10005   | 12/24/2003 | 1        | A       | 30  |             |
| 40001   | 1/9/2004   | 4        | A       | 40  |             |
| 10006   | 1/18/2004  | 1        | C       | 10  |             |
| 20001   | 2/12/2004  | 2        | B       | 20  | 20          |
| 40005   | 2/12/2004  | 4        | A       | 10  | 10          |
| 20002   | 2/16/2004  | 2        | C       | 20  | 20          |

| SALESID | DATEID    | SELLERID | BUYERID | QTY | QTY_SHIPPED |
|---------|-----------|----------|---------|-----|-------------|
| 30003   | 4/18/2004 | 3        | B       | 15  |             |
| 30004   | 4/18/2004 | 3        | B       | 20  |             |
| 30007   | 9/7/2004  | 3        | C       | 30  |             |

The following script creates and populates the sample WINSALES table.

```

CREATE TABLE winsales(
 salesid int,
 dateid date,
 sellerid int,
 buyerid char(10),
 qty int,
 qty_shipped int);

INSERT INTO winsales VALUES
 (30001, '8/2/2003', 3, 'b', 10, 10),
 (10001, '12/24/2003', 1, 'c', 10, 10),
 (10005, '12/24/2003', 1, 'a', 30, null),
 (40001, '1/9/2004', 4, 'a', 40, null),
 (10006, '1/18/2004', 1, 'c', 10, null),
 (20001, '2/12/2004', 2, 'b', 20, 20),
 (40005, '2/12/2004', 4, 'a', 10, 10),
 (20002, '2/16/2004', 2, 'c', 20, 20),
 (30003, '4/18/2004', 3, 'b', 15, null),
 (30004, '4/18/2004', 3, 'b', 20, null),
 (30007, '9/7/2004', 3, 'c', 30, null);

```

## AVG window function

The AVG window function returns the average (arithmetic mean) of the input expression values. The AVG function works with numeric values and ignores NULL values.

### Syntax

```

AVG ([ALL] expression) OVER
(
[PARTITION BY expr_list]
[ORDER BY order_list
 frame_clause]
)

```

### Arguments

*expression*

The target column or expression that the function operates on.

**ALL**

With the argument ALL, the function retains all duplicate values from the expression for counting. ALL is the default. DISTINCT is not supported.

**OVER**

Specifies the window clauses for the aggregation functions. The OVER clause distinguishes window aggregation functions from normal set aggregation functions.

PARTITION BY *expr\_list*

Defines the window for the AVG function in terms of one or more expressions.

ORDER BY *order\_list*

Sorts the rows within each partition. If no PARTITION BY is specified, ORDER BY uses the entire table.

*frame\_clause*

If an ORDER BY clause is used for an aggregate function, an explicit frame clause is required. The frame clause refines the set of rows in a function's window, including or excluding sets of rows within the ordered result. The frame clause consists of the ROWS keyword and associated specifiers. See [Window function syntax summary \(p. 1336\)](#).

## Data types

The argument types supported by the AVG function are SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL, and DOUBLE PRECISION.

The return types supported by the AVG function are:

- BIGINT for SMALLINT or INTEGER arguments
- NUMERIC for BIGINT arguments
- DOUBLE PRECISION for floating point arguments

## Examples

The following example computes a rolling average of quantities sold by date; order the results by date ID and sales ID:

```
select salesid, dateid, sellerid, qty,
avg(qty) over
(order by dateid, salesid rows unbounded preceding) as avg
from winsales
order by 2,1;

salesid | dateid | sellerid | qty | avg
-----+-----+-----+-----+
30001 | 2003-08-02 | 3 | 10 | 10
10001 | 2003-12-24 | 1 | 10 | 10
10005 | 2003-12-24 | 1 | 30 | 16
40001 | 2004-01-09 | 4 | 40 | 22
10006 | 2004-01-18 | 1 | 10 | 20
20001 | 2004-02-12 | 2 | 20 | 20
40005 | 2004-02-12 | 4 | 10 | 18
20002 | 2004-02-16 | 2 | 20 | 18
30003 | 2004-04-18 | 3 | 15 | 18
30004 | 2004-04-18 | 3 | 20 | 18
30007 | 2004-09-07 | 3 | 30 | 19
(11 rows)
```

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

## COUNT window function

The COUNT window function counts the rows defined by the expression.

The COUNT function has two variations. COUNT(\*) counts all the rows in the target table whether they include nulls or not. COUNT(expression) computes the number of rows with non-NULL values in a specific column or expression.

## Syntax

```
COUNT (* | [ALL] expression) OVER
(
[PARTITION BY expr_list]
[ORDER BY order_list
 frame_clause]
)
```

## Arguments

### *expression*

The target column or expression that the function operates on.

### ALL

With the argument ALL, the function retains all duplicate values from the expression for counting. ALL is the default. DISTINCT is not supported.

### OVER

Specifies the window clauses for the aggregation functions. The OVER clause distinguishes window aggregation functions from normal set aggregation functions.

### PARTITION BY *expr\_list*

Defines the window for the COUNT function in terms of one or more expressions.

### ORDER BY *order\_list*

Sorts the rows within each partition. If no PARTITION BY is specified, ORDER BY uses the entire table.

### *frame\_clause*

If an ORDER BY clause is used for an aggregate function, an explicit frame clause is required. The frame clause refines the set of rows in a function's window, including or excluding sets of rows within the ordered result. The frame clause consists of the ROWS keyword and associated specifiers. See [Window function syntax summary \(p. 1336\)](#).

## Data types

The COUNT function supports all argument data types.

The return type supported by the COUNT function is BIGINT.

## Examples

The following example shows the sales ID, quantity, and count of all rows from the beginning of the data window:

```
select salesid, qty,
count(*) over (order by salesid rows unbounded preceding) as count
from winsales
order by salesid;

salesid | qty | count
-----+----+-----

```

|           |    |    |
|-----------|----|----|
| 10001     | 10 | 1  |
| 10005     | 30 | 2  |
| 10006     | 10 | 3  |
| 20001     | 20 | 4  |
| 20002     | 20 | 5  |
| 30001     | 10 | 6  |
| 30003     | 15 | 7  |
| 30004     | 20 | 8  |
| 30007     | 30 | 9  |
| 40001     | 40 | 10 |
| 40005     | 10 | 11 |
| (11 rows) |    |    |

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

The following example shows how the sales ID, quantity, and count of non-null rows from the beginning of the data window. (In the WINSALES table, the QTY\_SHIPPED column contains some NULLs.)

```
select salesid, qty, qty_shipped,
count(qty_shipped)
over (order by salesid rows unbounded preceding) as count
from winsales
order by salesid;

salesid | qty | qty_shipped | count
-----+----+-----+-----
10001 | 10 | 10 | 1
10005 | 30 | | 1
10006 | 10 | | 1
20001 | 20 | 20 | 2
20002 | 20 | 20 | 3
30001 | 10 | 10 | 4
30003 | 15 | | 4
30004 | 20 | | 4
30007 | 30 | | 4
40001 | 40 | | 4
40005 | 10 | 10 | 5
(11 rows)
```

## CUME\_DIST window function

Calculates the cumulative distribution of a value within a window or partition. Assuming ascending ordering, the cumulative distribution is determined using this formula:

count of rows with values  $\leq x$  / count of rows in the window or partition

where  $x$  equals the value in the current row of the column specified in the ORDER BY clause. The following dataset illustrates use of this formula:

| Row# | Value | Calculation | CUME_DIST |
|------|-------|-------------|-----------|
| 1    | 2500  | (1)/(5)     | 0.2       |
| 2    | 2600  | (2)/(5)     | 0.4       |
| 3    | 2800  | (3)/(5)     | 0.6       |
| 4    | 2900  | (4)/(5)     | 0.8       |
| 5    | 3100  | (5)/(5)     | 1.0       |

The return value range is >0 to 1, inclusive.

## Syntax

|              |
|--------------|
| CUME_DIST () |
|--------------|

```
OVER (
 [PARTITION BY partition_expression]
 [ORDER BY order_list]
)
```

## Arguments

### OVER

A clause that specifies the window partitioning. The OVER clause cannot contain a window frame specification.

### PARTITION BY *partition\_expression*

Optional. An expression that sets the range of records for each group in the OVER clause.

### ORDER BY *order\_list*

The expression on which to calculate cumulative distribution. The expression must have either a numeric data type or be implicitly convertible to one. If ORDER BY is omitted, the return value is 1 for all rows.

If ORDER BY doesn't produce a unique ordering, the order of the rows is nondeterministic. For more information, see [Unique ordering of data for window functions \(p. 1339\)](#).

## Return type

FLOAT8

## Examples

The following example calculates the cumulative distribution of the quantity for each seller:

```
select sellerid, qty, cume_dist()
over (partition by sellerid order by qty)
from winsales;

sellerid qty cume_dist

1 10.00 0.33
1 10.64 0.67
1 30.37 1
3 10.04 0.25
3 15.15 0.5
3 20.75 0.75
3 30.55 1
2 20.09 0.5
2 20.12 1
4 10.12 0.5
4 40.23 1
```

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

## DENSE\_RANK window function

The DENSE\_RANK window function determines the rank of a value in a group of values, based on the ORDER BY expression in the OVER clause. If the optional PARTITION BY clause is present, the rankings are reset for each group of rows. Rows with equal values for the ranking criteria receive the same rank. The DENSE\_RANK function differs from RANK in one respect: If two or more rows tie, there is no gap in the sequence of ranked values. For example, if two rows are ranked 1, the next rank is 2.

You can have ranking functions with different PARTITION BY and ORDER BY clauses in the same query.

## Syntax

```
DENSE_RANK () OVER
(
[PARTITION BY expr_list]
[ORDER BY order_list]
)
```

## Arguments

()

The function takes no arguments, but the empty parentheses are required.

OVER

The window clauses for the DENSE\_RANK function.

PARTITION BY *expr\_list*

Optional. One or more expressions that define the window.

ORDER BY *order\_list*

Optional. The expression on which the ranking values are based. If no PARTITION BY is specified, ORDER BY uses the entire table. If ORDER BY is omitted, the return value is 1 for all rows.

If ORDER BY doesn't produce a unique ordering, the order of the rows is nondeterministic. For more information, see [Unique ordering of data for window functions \(p. 1339\)](#).

## Return type

INTEGER

## Examples

The following example orders the table by the quantity sold (in descending order), and assign both a dense rank and a regular rank to each row. The results are sorted after the window function results are applied.

```
select salesid, qty,
dense_rank() over(order by qty desc) as d_rnk,
rank() over(order by qty desc) as rnk
from winsales
order by 2,1;

salesid | qty | d_rnk | rnk
-----+----+-----+-----
10001 | 10 | 5 | 8
10006 | 10 | 5 | 8
30001 | 10 | 5 | 8
40005 | 10 | 5 | 8
30003 | 15 | 4 | 7
20001 | 20 | 3 | 4
20002 | 20 | 3 | 4
30004 | 20 | 3 | 4
10005 | 30 | 2 | 2
30007 | 30 | 2 | 2
40001 | 40 | 1 | 1
(11 rows)
```

Note the difference in rankings assigned to the same set of rows when the DENSE\_RANK and RANK functions are used side by side in the same query. For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

The following example partitions the table by SELLERID and orders each partition by the quantity (in descending order) and assign a dense rank to each row. The results are sorted after the window function results are applied.

```
select salesid, sellerid, qty,
dense_rank() over(partition by sellerid order by qty desc) as d_rnk
from winsales
order by 2,3,1;

salesid | sellerid | qty | d_rnk
-----+-----+-----+
10001 | 1 | 10 | 2
10006 | 1 | 10 | 2
10005 | 1 | 30 | 1
20001 | 2 | 20 | 1
20002 | 2 | 20 | 1
30001 | 3 | 10 | 4
30003 | 3 | 15 | 3
30004 | 3 | 20 | 2
30007 | 3 | 30 | 1
40005 | 4 | 10 | 2
40001 | 4 | 40 | 1
(11 rows)
```

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

## FIRST\_VALUE window function

Given an ordered set of rows, FIRST\_VALUE returns the value of the specified expression with respect to the first row in the window frame.

For information about selecting the last row in the frame, see [LAST\\_VALUE window function \(p. 1351\)](#).

### Syntax

```
FIRST_VALUE(expression)[IGNORE NULLS | RESPECT NULLS]
OVER (
[PARTITION BY expr_list]
[ORDER BY order_list frame_clause]
)
```

### Arguments

#### *expression*

The target column or expression that the function operates on.

#### IGNORE NULLS

When this option is used with FIRST\_VALUE, the function returns the first value in the frame that is not NULL (or NULL if all values are NULL).

#### RESPECT NULLS

Indicates that Amazon Redshift should include null values in the determination of which row to use. RESPECT NULLS is supported by default if you do not specify IGNORE NULLS.

## OVER

Introduces the window clauses for the function.

### PARTITION BY *expr\_list*

Defines the window for the function in terms of one or more expressions.

### ORDER BY *order\_list*

Sorts the rows within each partition. If no PARTITION BY clause is specified, ORDER BY sorts the entire table. If you specify an ORDER BY clause, you must also specify a *frame\_clause*.

The results of the FIRST\_VALUE function depends on the ordering of the data. The results are nondeterministic in the following cases:

- When no ORDER BY clause is specified and a partition contains two different values for an expression
- When the expression evaluates to different values that correspond to the same value in the ORDER BY list.

### *frame\_clause*

If an ORDER BY clause is used for an aggregate function, an explicit frame clause is required. The frame clause refines the set of rows in a function's window, including or excluding sets of rows in the ordered result. The frame clause consists of the ROWS keyword and associated specifiers. See [Window function syntax summary \(p. 1336\)](#).

## Return type

These functions support expressions that use primitive Amazon Redshift data types. The return type is the same as the data type of the *expression*.

## Examples

The following examples use the VENUE table from the sample TICKIT data. For more information, see [Sample database \(p. 1704\)](#).

The following example returns the seating capacity for each venue in the VENUE table, with the results ordered by capacity (high to low). The FIRST\_VALUE function is used to select the name of the venue that corresponds to the first row in the frame: in this case, the row with the highest number of seats. The results are partitioned by state, so when the VENUESTATE value changes, a new first value is selected. The window frame is unbounded so the same first value is selected for each row in each partition.

For California, Qualcomm Stadium has the highest number of seats (70561), so this name is the first value for all of the rows in the CA partition.

```
select venuestate, venues Seats, venuename,
first_value(venuename)
over(partition by venuestate
order by venues desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venues >0)
order by venuestate;
```

| venuestate | venues | venuename        | first_value      |
|------------|--------|------------------|------------------|
| CA         | 70561  | Qualcomm Stadium | Qualcomm Stadium |
| CA         | 69843  | Monster Park     | Qualcomm Stadium |
| CA         | 63026  | McAfee Coliseum  | Qualcomm Stadium |
| CA         | 56000  | Dodger Stadium   | Qualcomm Stadium |

|     |       |                                |                  |
|-----|-------|--------------------------------|------------------|
| CA  | 45050 | Angel Stadium of Anaheim       | Qualcomm Stadium |
| CA  | 42445 | PETCO Park                     | Qualcomm Stadium |
| CA  | 41503 | AT&T Park                      | Qualcomm Stadium |
| CA  | 22000 | Shoreline Amphitheatre         | Qualcomm Stadium |
| CO  | 76125 | INVESCO Field                  | INVESCO Field    |
| CO  | 50445 | Coors Field                    | INVESCO Field    |
| DC  | 41888 | Nationals Park                 | Nationals Park   |
| FL  | 74916 | Dolphin Stadium                | Dolphin Stadium  |
| FL  | 73800 | Jacksonville Municipal Stadium | Dolphin Stadium  |
| FL  | 65647 | Raymond James Stadium          | Dolphin Stadium  |
| FL  | 36048 | Tropicana Field                | Dolphin Stadium  |
| ... |       |                                |                  |

The following example shows the use of the IGNORE NULLS option and relies on the addition of a new row to the VENUE table:

```
insert into venue values(2000,null,'Stanford','CA',90000);
```

This new row contains a NULL value for the VENUENAME column. Now repeat the FIRST\_VALUE query that was shown earlier in this section:

```
select venuestate, venues seats, venuename,
first_value(venuename)
over(partition by venuestate
order by venues seats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venues seats >0)
order by venuestate;
```

| venuestate | venues seats | venuename        | first_value |
|------------|--------------|------------------|-------------|
| CA         | 90000        | NULL             | NULL        |
| CA         | 70561        | Qualcomm Stadium | NULL        |
| CA         | 69843        | Monster Park     | NULL        |
| ...        |              |                  |             |

Because the new row contains the highest VENUSEATS value (90000) and its VENUENAME is NULL, the FIRST\_VALUE function returns NULL for the CA partition. To ignore rows like this in the function evaluation, add the IGNORE NULLS option to the function argument:

```
select venuestate, venues seats, venuename,
first_value(venuename) ignore nulls
over(partition by venuestate
order by venues seats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venuestate='CA')
order by venuestate;
```

| venuestate | venues seats | venuename        | first_value      |
|------------|--------------|------------------|------------------|
| CA         | 90000        | NULL             | Qualcomm Stadium |
| CA         | 70561        | Qualcomm Stadium | Qualcomm Stadium |
| CA         | 69843        | Monster Park     | Qualcomm Stadium |
| ...        |              |                  |                  |

## LAG window function

The LAG window function returns the values for a row at a given offset above (before) the current row in the partition.

## Syntax

```
LAG (value_expr [, offset])
[IGNORE NULLS | RESPECT NULLS]
OVER ([PARTITION BY window_partition] ORDER BY window_ordering)
```

## Arguments

### *value\_expr*

The target column or expression that the function operates on.

### *offset*

An optional parameter that specifies the number of rows before the current row to return values for. The offset can be a constant integer or an expression that evaluates to an integer. If you do not specify an offset, Amazon Redshift uses 1 as the default value. An offset of 0 indicates the current row.

### IGNORE NULLS

An optional specification that indicates that Amazon Redshift should skip null values in the determination of which row to use. Null values are included if IGNORE NULLS is not listed.

#### Note

You can use an NVL or COALESCE expression to replace the null values with another value.  
For more information, see [NVL function \(p. 1061\)](#).

### RESPECT NULLS

Indicates that Amazon Redshift should include null values in the determination of which row to use. RESPECT NULLS is supported by default if you do not specify IGNORE NULLS.

### OVER

Specifies the window partitioning and ordering. The OVER clause cannot contain a window frame specification.

### PARTITION BY *window\_partition*

An optional argument that sets the range of records for each group in the OVER clause.

### ORDER BY *window\_ordering*

Sorts the rows within each partition.

The LAG window function supports expressions that use any of the Amazon Redshift data types. The return type is the same as the type of the *value\_expr*.

## Examples

The following example shows the quantity of tickets sold to the buyer with a buyer ID of 3 and the time that buyer 3 bought the tickets. To compare each sale with the previous sale for buyer 3, the query returns the previous quantity sold for each sale. Since there is no purchase before 1/16/2008, the first previous quantity sold value is null:

```
select buyerid, saletime, qtysold,
lag(qtysold,1) over (order by buyerid, saletime) as prev_qtysold
from sales where buyerid = 3 order by buyerid, saletime;

buyerid | saletime | qtysold | prev_qtysold
```

| 3         | 2008-01-16 01:06:09 | 1 |  |   |
|-----------|---------------------|---|--|---|
| 3         | 2008-01-28 02:10:01 | 1 |  | 1 |
| 3         | 2008-03-12 10:39:53 | 1 |  | 1 |
| 3         | 2008-03-13 02:56:07 | 1 |  | 1 |
| 3         | 2008-03-29 08:21:39 | 2 |  | 1 |
| 3         | 2008-04-27 02:39:01 | 1 |  | 2 |
| 3         | 2008-08-16 07:04:37 | 2 |  | 1 |
| 3         | 2008-08-22 11:45:26 | 2 |  | 2 |
| 3         | 2008-09-12 09:11:25 | 1 |  | 2 |
| 3         | 2008-10-01 06:22:37 | 1 |  | 1 |
| 3         | 2008-10-20 01:55:51 | 2 |  | 1 |
| 3         | 2008-10-28 01:30:40 | 1 |  | 2 |
| (12 rows) |                     |   |  |   |

## LAST\_VALUE window function

Given an ordered set of rows, The LAST\_VALUE function returns the value of the expression with respect to the last row in the frame.

For information about selecting the first row in the frame, see [FIRST\\_VALUE window function \(p. 1347\)](#).

### Syntax

```
LAST_VALUE(expression)[IGNORE NULLS | RESPECT NULLS]
OVER (
[PARTITION BY expr_list]
[ORDER BY order_list frame_clause]
)
```

### Arguments

#### *expression*

The target column or expression that the function operates on.

#### IGNORE NULLS

The function returns the last value in the frame that is not NULL (or NULL if all values are NULL).

#### RESPECT NULLS

Indicates that Amazon Redshift should include null values in the determination of which row to use. RESPECT NULLS is supported by default if you do not specify IGNORE NULLS.

#### OVER

Introduces the window clauses for the function.

#### PARTITION BY *expr\_list*

Defines the window for the function in terms of one or more expressions.

#### ORDER BY *order\_list*

Sorts the rows within each partition. If no PARTITION BY clause is specified, ORDER BY sorts the entire table. If you specify an ORDER BY clause, you must also specify a *frame\_clause*.

The results depend on the ordering of the data. The results are nondeterministic in the following cases:

- When no ORDER BY clause is specified and a partition contains two different values for an expression

- When the expression evaluates to different values that correspond to the same value in the ORDER BY list.

#### *frame\_clause*

If an ORDER BY clause is used for an aggregate function, an explicit frame clause is required. The frame clause refines the set of rows in a function's window, including or excluding sets of rows in the ordered result. The frame clause consists of the ROWS keyword and associated specifiers. See [Window function syntax summary \(p. 1336\)](#).

## Return type

These functions support expressions that use primitive Amazon Redshift data types. The return type is the same as the data type of the *expression*.

## Examples

The following examples use the VENUE table from the sample TICKIT data. For more information, see [Sample database \(p. 1704\)](#).

The following example returns the seating capacity for each venue in the VENUE table, with the results ordered by capacity (high to low). The LAST\_VALUE function is used to select the name of the venue that corresponds to the last row in the frame: in this case, the row with the least number of seats. The results are partitioned by state, so when the VENUESTATE value changes, a new last value is selected. The window frame is unbounded so the same last value is selected for each row in each partition.

For California, Shoreline Amphitheatre is returned for every row in the partition because it has the lowest number of seats (22000).

```
select venuestate, venueseats, venuename,
last_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;
```

| venuestate | venueseats | venuename                      | last_value             |
|------------|------------|--------------------------------|------------------------|
| CA         | 70561      | Qualcomm Stadium               | Shoreline Amphitheatre |
| CA         | 69843      | Monster Park                   | Shoreline Amphitheatre |
| CA         | 63026      | McAfee Coliseum                | Shoreline Amphitheatre |
| CA         | 56000      | Dodger Stadium                 | Shoreline Amphitheatre |
| CA         | 45050      | Angel Stadium of Anaheim       | Shoreline Amphitheatre |
| CA         | 42445      | PETCO Park                     | Shoreline Amphitheatre |
| CA         | 41503      | AT&T Park                      | Shoreline Amphitheatre |
| CA         | 22000      | Shoreline Amphitheatre         | Shoreline Amphitheatre |
| CO         | 76125      | INVESCO Field                  | Coors Field            |
| CO         | 50445      | Coors Field                    | Coors Field            |
| DC         | 41888      | Nationals Park                 | Nationals Park         |
| FL         | 74916      | Dolphin Stadium                | Tropicana Field        |
| FL         | 73800      | Jacksonville Municipal Stadium | Tropicana Field        |
| FL         | 65647      | Raymond James Stadium          | Tropicana Field        |
| FL         | 36048      | Tropicana Field                | Tropicana Field        |
| ...        |            |                                |                        |

## LEAD window function

The LEAD window function returns the values for a row at a given offset below (after) the current row in the partition.

## Syntax

```
LEAD (value_expr [, offset])
[IGNORE NULLS | RESPECT NULLS]
OVER ([PARTITION BY window_partition] ORDER BY window_ordering)
```

## Arguments

### *value\_expr*

The target column or expression that the function operates on.

### *offset*

An optional parameter that specifies the number of rows below the current row to return values for. The offset can be a constant integer or an expression that evaluates to an integer. If you do not specify an offset, Amazon Redshift uses 1 as the default value. An offset of 0 indicates the current row.

### IGNORE NULLS

An optional specification that indicates that Amazon Redshift should skip null values in the determination of which row to use. Null values are included if IGNORE NULLS is not listed.

#### Note

You can use an NVL or COALESCE expression to replace the null values with another value. For more information, see [NVL function \(p. 1061\)](#).

### RESPECT NULLS

Indicates that Amazon Redshift should include null values in the determination of which row to use. RESPECT NULLS is supported by default if you do not specify IGNORE NULLS.

### OVER

Specifies the window partitioning and ordering. The OVER clause cannot contain a window frame specification.

### PARTITION BY *window\_partition*

An optional argument that sets the range of records for each group in the OVER clause.

### ORDER BY *window\_ordering*

Sorts the rows within each partition.

The LEAD window function supports expressions that use any of the Amazon Redshift data types. The return type is the same as the type of the *value\_expr*.

## Examples

The following example provides the commission for events in the SALES table for which tickets were sold on January 1, 2008 and January 2, 2008 and the commission paid for ticket sales for the subsequent sale.

```
select eventid, commission, saletime,
lead(commission, 1) over (order by saletime) as next_comm
from sales where saletime between '2008-01-01 00:00:00' and '2008-01-02 12:59:59'
order by saletime;
```

| eventid | commission | saletime            | next_comm |
|---------|------------|---------------------|-----------|
| 6213    | 52.05      | 2008-01-01 01:00:19 | 106.20    |

|      |           |                     |         |
|------|-----------|---------------------|---------|
| 7003 | 106.20    | 2008-01-01 02:30:52 | 103.20  |
| 8762 | 103.20    | 2008-01-01 03:50:02 | 70.80   |
| 1150 | 70.80     | 2008-01-01 06:06:57 | 50.55   |
| 1749 | 50.55     | 2008-01-01 07:05:02 | 125.40  |
| 8649 | 125.40    | 2008-01-01 07:26:20 | 35.10   |
| 2903 | 35.10     | 2008-01-01 09:41:06 | 259.50  |
| 6605 | 259.50    | 2008-01-01 12:50:55 | 628.80  |
| 6870 | 628.80    | 2008-01-01 12:59:34 | 74.10   |
| 6977 | 74.10     | 2008-01-02 01:11:16 | 13.50   |
| 4650 | 13.50     | 2008-01-02 01:40:59 | 26.55   |
| 4515 | 26.55     | 2008-01-02 01:52:35 | 22.80   |
| 5465 | 22.80     | 2008-01-02 02:28:01 | 45.60   |
| 5465 | 45.60     | 2008-01-02 02:28:02 | 53.10   |
| 7003 | 53.10     | 2008-01-02 02:31:12 | 70.35   |
| 4124 | 70.35     | 2008-01-02 03:12:50 | 36.15   |
| 1673 | 36.15     | 2008-01-02 03:15:00 | 1300.80 |
| ...  |           |                     |         |
|      | (39 rows) |                     |         |

## LISTAGG window function

For each group in a query, the LISTAGG window function orders the rows for that group according to the ORDER BY expression, then concatenates the values into a single string.

LISTAGG is a compute-node only function. The function returns an error if the query doesn't reference a user-defined table or Amazon Redshift system table. For more information, see [Querying the catalog tables \(p. 1680\)](#).

### Syntax

```
LISTAGG([DISTINCT] expression [, 'delimiter'])
[WITHIN GROUP (ORDER BY order_list)]
OVER ([PARTITION BY partition_expression])
```

### Arguments

#### DISTINCT

(Optional) A clause that eliminates duplicate values from the specified expression before concatenating. Trailing spaces are ignored, so the strings 'a ' and 'a' are treated as duplicates. LISTAGG uses the first value encountered. For more information, see [Significance of trailing blanks \(p. 584\)](#).

#### aggregate\_expression

Any valid expression (such as a column name) that provides the values to aggregate. NULL values and empty strings are ignored.

#### delimiter

(Optional) The string constant to separate the concatenated values. The default is NULL.

#### WITHIN GROUP (ORDER BY order\_list)

(Optional) A clause that specifies the sort order of the aggregated values. Deterministic only if ORDER BY provides unique ordering. The default is to aggregate all rows and return a single value.

#### OVER

A clause that specifies the window partitioning. The OVER clause cannot contain a window ordering or window frame specification.

PARTITION BY *partition\_expression*

(Optional) Sets the range of records for each group in the OVER clause.

## Returns

VARCHAR(MAX). If the result set is larger than the maximum VARCHAR size (64K – 1, or 65535), then LISTAGG returns the following error:

```
Invalid operation: Result size exceeds LISTAGG limit
```

## Examples

The following examples uses the WINSALES table. For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

The following example returns a list of seller IDs, ordered by seller ID.

```
select listagg(sellerid)
within group (order by sellerid)
over() from winsales;

listagg

11122333344
...
...
11122333344
11122333344
(11 rows)
```

The following example returns a list of seller IDs for buyer B, ordered by date.

```
select listagg(sellerid)
within group (order by dateid)
over () as seller
from winsales
where buyerid = 'b' ;

seller

3233
3233
3233
3233

(4 rows)
```

The following example returns a comma-separated list of sales dates for buyer B.

```
select listagg(dateid,',')
within group (order by sellerid desc,salesid asc)
over () as dates
from winsales
where buyerid = 'b';

dates

```

```

2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12

(4 rows)

```

The following example uses DISTINCT to return a list of unique sales dates for buyer B.

```

select listagg(distinct dateid,',')
within group (order by sellerid desc,salesid asc)
over () as dates
from winsales
where buyerid = 'b';

 dates

2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12

(4 rows)

```

The following example returns a comma-separated list of sales IDs for each buyer ID.

```

select buyerid,
listagg(salesid,',')
within group (order by salesid)
over (partition by buyerid) as sales_id
from winsales
order by buyerid;

 buyerid | sales_id
 +-----+
 a |10005,40001,40005
 a |10005,40001,40005
 a |10005,40001,40005
 b |20001,30001,30004,30003
 b |20001,30001,30004,30003
 b |20001,30001,30004,30003
 b |20001,30001,30004,30003
 c |10001,20002,30007,10006
 c |10001,20002,30007,10006
 c |10001,20002,30007,10006
 c |10001,20002,30007,10006

(11 rows)

```

## MAX window function

The MAX window function returns the maximum of the input expression values. The MAX function works with numeric values and ignores NULL values.

### Syntax

```

MAX ([ALL] expression) OVER
(
[PARTITION BY expr_list]
[ORDER BY order_list frame_clause]
)

```

## Arguments

### *expression*

The target column or expression that the function operates on.

### ALL

With the argument ALL, the function retains all duplicate values from the expression. ALL is the default. DISTINCT is not supported.

### OVER

A clause that specifies the window clauses for the aggregation functions. The OVER clause distinguishes window aggregation functions from normal set aggregation functions.

### PARTITION BY *expr\_list*

Defines the window for the MAX function in terms of one or more expressions.

### ORDER BY *order\_list*

Sorts the rows within each partition. If no PARTITION BY is specified, ORDER BY uses the entire table.

### *frame\_clause*

If an ORDER BY clause is used for an aggregate function, an explicit frame clause is required. The frame clause refines the set of rows in a function's window, including or excluding sets of rows within the ordered result. The frame clause consists of the ROWS keyword and associated specifiers. See [Window function syntax summary \(p. 1336\)](#).

## Data types

Accepts any data type as input. Returns the same data type as *expression*.

## Examples

The following example shows the sales ID, quantity, and maximum quantity from the beginning of the data window:

```
select salesid, qty,
max(qty) over (order by salesid rows unbounded preceding) as max
from winsales
order by salesid;

salesid | qty | max
-----+----+-----
10001 | 10 | 10
10005 | 30 | 30
10006 | 10 | 30
20001 | 20 | 30
20002 | 20 | 30
30001 | 10 | 30
30003 | 15 | 30
30004 | 20 | 30
30007 | 30 | 30
40001 | 40 | 40
40005 | 10 | 40
(11 rows)
```

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

The following example shows the salesid, quantity, and maximum quantity in a restricted frame:

```
select salesid, qty,
max(qty) over (order by salesid rows between 2 preceding and 1 preceding) as max
from winsales
order by salesid;

salesid | qty | max
-----+----+-----
10001 | 10 |
10005 | 30 | 10
10006 | 10 | 30
20001 | 20 | 30
20002 | 20 | 20
30001 | 10 | 20
30003 | 15 | 20
30004 | 20 | 15
30007 | 30 | 20
40001 | 40 | 30
40005 | 10 | 40
(11 rows)
```

## MEDIAN window function

Calculates the median value for the range of values in a window or partition. NULL values in the range are ignored.

MEDIAN is an inverse distribution function that assumes a continuous distribution model.

MEDIAN is a compute-node only function. The function returns an error if the query doesn't reference a user-defined table or Amazon Redshift system table.

### Syntax

```
MEDIAN (median_expression)
OVER ([PARTITION BY partition_expression])
```

### Arguments

#### *median\_expression*

An expression, such as a column name, that provides the values for which to determine the median. The expression must have either a numeric or datetime data type or be implicitly convertible to one.

#### OVER

A clause that specifies the window partitioning. The OVER clause cannot contain a window ordering or window frame specification.

#### PARTITION BY *partition\_expression*

Optional. An expression that sets the range of records for each group in the OVER clause.

### Data types

The return type is determined by the data type of *median\_expression*. The following table shows the return type for each *median\_expression* data type.

| Input Type                         | Return Type |
|------------------------------------|-------------|
| INT2, INT4, INT8, NUMERIC, DECIMAL | DECIMAL     |

| Input Type    | Return Type |
|---------------|-------------|
| FLOAT, DOUBLE | DOUBLE      |
| DATE          | DATE        |

## Usage notes

If the *median\_expression* argument is a DECIMAL data type defined with the maximum precision of 38 digits, it is possible that MEDIAN will return either an inaccurate result or an error. If the return value of the MEDIAN function exceeds 38 digits, the result is truncated to fit, which causes a loss of precision. If, during interpolation, an intermediate result exceeds the maximum precision, a numeric overflow occurs and the function returns an error. To avoid these conditions, we recommend either using a data type with lower precision or casting the *median\_expression* argument to a lower precision.

For example, a SUM function with a DECIMAL argument returns a default precision of 38 digits. The scale of the result is the same as the scale of the argument. So, for example, a SUM of a DECIMAL(5,2) column returns a DECIMAL(38,2) data type.

The following example uses a SUM function in the *median\_expression* argument of a MEDIAN function. The data type of the PRICEPAID column is DECIMAL (8,2), so the SUM function returns DECIMAL(38,2).

```
select salesid, sum(pricepaid), median(sum(pricepaid))
over() from sales where salesid < 10 group by salesid;
```

To avoid a potential loss of precision or an overflow error, cast the result to a DECIMAL data type with lower precision, as the following example shows.

```
select salesid, sum(pricepaid), median(sum(pricepaid))::decimal(30,2)
over() from sales where salesid < 10 group by salesid;
```

## Examples

The following example calculates the median sales quantity for each seller:

```
select sellerid, qty, median(qty)
over (partition by sellerid)
from winsales
order by sellerid;

sellerid qty median

1 10 10.0
1 10 10.0
1 30 10.0
2 20 20.0
2 20 20.0
3 10 17.5
3 15 17.5
3 20 17.5
3 30 17.5
4 10 25.0
4 40 25.0
```

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

## MIN window function

The MIN window function returns the minimum of the input expression values. The MIN function works with numeric values and ignores NULL values.

### Syntax

```
MIN ([ALL] expression) OVER
(
[PARTITION BY expr_list]
[ORDER BY order_list frame_clause]
)
```

### Arguments

#### *expression*

The target column or expression that the function operates on.

#### ALL

With the argument ALL, the function retains all duplicate values from the expression. ALL is the default. DISTINCT is not supported.

#### OVER

Specifies the window clauses for the aggregation functions. The OVER clause distinguishes window aggregation functions from normal set aggregation functions.

#### PARTITION BY *expr\_list*

Defines the window for the MIN function in terms of one or more expressions.

#### ORDER BY *order\_list*

Sorts the rows within each partition. If no PARTITION BY is specified, ORDER BY uses the entire table.

#### *frame\_clause*

If an ORDER BY clause is used for an aggregate function, an explicit frame clause is required. The frame clause refines the set of rows in a function's window, including or excluding sets of rows within the ordered result. The frame clause consists of the ROWS keyword and associated specifiers. See [Window function syntax summary \(p. 1336\)](#).

### Data types

Accepts any data type as input. Returns the same data type as *expression*.

### Examples

The following example shows the sales ID, quantity, and minimum quantity from the beginning of the data window:

```
select salesid, qty,
min(qty) over
(order by salesid rows unbounded preceding)
from winsales
order by salesid;
```

| salesid   | qty | min |
|-----------|-----|-----|
| 10001     | 10  | 10  |
| 10005     | 30  | 10  |
| 10006     | 10  | 10  |
| 20001     | 20  | 10  |
| 20002     | 20  | 10  |
| 30001     | 10  | 10  |
| 30003     | 15  | 10  |
| 30004     | 20  | 10  |
| 30007     | 30  | 10  |
| 40001     | 40  | 10  |
| 40005     | 10  | 10  |
| (11 rows) |     |     |

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

The following example shows the sales ID, quantity, and minimum quantity in a restricted frame:

```
select salesid, qty,
min(qty) over
(order by salesid rows between 2 preceding and 1 preceding) as min
from winsales
order by salesid;

salesid	qty	min
10001	10	
10005	30	10
10006	10	10
20001	20	10
20002	20	10
30001	10	20
30003	15	10
30004	20	10
30007	30	15
40001	40	20
40005	10	30
(11 rows)		


```

## NTH\_VALUE window function

The NTH\_VALUE window function returns the expression value of the specified row of the window frame relative to the first row of the window.

### Syntax

```
NTH_VALUE (expr, offset)
[IGNORE NULLS | RESPECT NULLS]
OVER
([PARTITION BY window_partition]
[ORDER BY window_ordering
 frame_clause])
```

### Arguments

*expr*

The target column or expression that the function operates on.

#### *offset*

Determines the row number relative to the first row in the window for which to return the expression. The *offset* can be a constant or an expression and must be a positive integer that is greater than 0.

#### IGNORE NULLS

An optional specification that indicates that Amazon Redshift should skip null values in the determination of which row to use. Null values are included if IGNORE NULLS is not listed.

#### RESPECT NULLS

Indicates that Amazon Redshift should include null values in the determination of which row to use. RESPECT NULLS is supported by default if you do not specify IGNORE NULLS.

#### OVER

Specifies the window partitioning, ordering, and window frame.

##### PARTITION BY *window\_partition*

Sets the range of records for each group in the OVER clause.

##### ORDER BY *window\_ordering*

Sorts the rows within each partition. If ORDER BY is omitted, the default frame consists of all rows in the partition.

##### *frame\_clause*

If an ORDER BY clause is used for an aggregate function, an explicit frame clause is required. The frame clause refines the set of rows in a function's window, including or excluding sets of rows in the ordered result. The frame clause consists of the ROWS keyword and associated specifiers. See [Window function syntax summary \(p. 1336\)](#).

The NTH\_VALUE window function supports expressions that use any of the Amazon Redshift data types. The return type is the same as the type of the *expr*.

## Examples

The following example shows the number of seats in the third largest venue in California, Florida, and New York compared to the number of seats in the other venues in those states:

```
select venuestate, venuename, venueseats,
nth_value(venueseats, 3)
ignore nulls
over(partition by venuestate order by venueseats desc
rows between unbounded preceding and unbounded following)
as third_most_seats
from (select * from venue where venueseats > 0 and
venuestate in('CA', 'FL', 'NY'))
order by venuestate;
```

| venuestate | venuename                | venueseats | third_most_seats |
|------------|--------------------------|------------|------------------|
| CA         | Qualcomm Stadium         | 70561      | 63026            |
| CA         | Monster Park             | 69843      | 63026            |
| CA         | McAfee Coliseum          | 63026      | 63026            |
| CA         | Dodger Stadium           | 56000      | 63026            |
| CA         | Angel Stadium of Anaheim | 45050      | 63026            |
| CA         | PETCO Park               | 42445      | 63026            |
| CA         | AT&T Park                | 41503      | 63026            |
| CA         | Shoreline Amphitheatre   | 22000      | 63026            |

|           |                                |       |       |
|-----------|--------------------------------|-------|-------|
| FL        | Dolphin Stadium                | 74916 | 65647 |
| FL        | Jacksonville Municipal Stadium | 73800 | 65647 |
| FL        | Raymond James Stadium          | 65647 | 65647 |
| FL        | Tropicana Field                | 36048 | 65647 |
| NY        | Ralph Wilson Stadium           | 73967 | 20000 |
| NY        | Yankee Stadium                 | 52325 | 20000 |
| NY        | Madison Square Garden          | 20000 | 20000 |
| (15 rows) |                                |       |       |

## NTILE window function

The NTILE window function divides ordered rows in the partition into the specified number of ranked groups of as equal size as possible and returns the group that a given row falls into.

### Syntax

```
NTILE (expr)
OVER (
[PARTITION BY expression_list]
[ORDER BY order_list]
)
```

### Arguments

#### *expr*

The number of ranking groups and must result in a positive integer value (greater than 0) for each partition. The *expr* argument must not be nullable.

#### OVER

A clause that specifies the window partitioning and ordering. The OVER clause cannot contain a window frame specification.

#### PARTITION BY *window\_partition*

Optional. The range of records for each group in the OVER clause.

#### ORDER BY *window\_ordering*

Optional. An expression that sorts the rows within each partition. If the ORDER BY clause is omitted, the ranking behavior is the same.

If ORDER BY does not produce a unique ordering, the order of the rows is nondeterministic. For more information, see [Unique ordering of data for window functions \(p. 1339\)](#).

### Return type

BIGINT

### Examples

The following example ranks into four ranking groups the price paid for Hamlet tickets on August 26, 2008. The result set is 17 rows, divided almost evenly among the rankings 1 through 4:

```
select eventname, caldate, pricepaid, ntile(4)
over(order by pricepaid desc) from sales, event, date
where sales.eventid=event.eventid and event.dateid=date.dateid and eventname='Hamlet'
and caldate='2008-08-26'
```

```
order by 4;

eventname | caldate | pricepaid | ntile
-----+-----+-----+-----
Hamlet | 2008-08-26 | 1883.00 | 1
Hamlet | 2008-08-26 | 1065.00 | 1
Hamlet | 2008-08-26 | 589.00 | 1
Hamlet | 2008-08-26 | 530.00 | 1
Hamlet | 2008-08-26 | 472.00 | 1
Hamlet | 2008-08-26 | 460.00 | 2
Hamlet | 2008-08-26 | 355.00 | 2
Hamlet | 2008-08-26 | 334.00 | 2
Hamlet | 2008-08-26 | 296.00 | 2
Hamlet | 2008-08-26 | 230.00 | 3
Hamlet | 2008-08-26 | 216.00 | 3
Hamlet | 2008-08-26 | 212.00 | 3
Hamlet | 2008-08-26 | 106.00 | 3
Hamlet | 2008-08-26 | 100.00 | 4
Hamlet | 2008-08-26 | 94.00 | 4
Hamlet | 2008-08-26 | 53.00 | 4
Hamlet | 2008-08-26 | 25.00 | 4
(17 rows)
```

## PERCENT\_RANK window function

Calculates the percent rank of a given row. The percent rank is determined using this formula:

$$(x - 1) / (\text{the number of rows in the window or partition} - 1)$$

where x is the rank of the current row. The following dataset illustrates use of this formula:

| Row# | Value | Rank | Calculation | PERCENT_RANK |
|------|-------|------|-------------|--------------|
| 1    | 15    | 1    | (1-1)/(7-1) | 0.0000       |
| 2    | 20    | 2    | (2-1)/(7-1) | 0.1666       |
| 3    | 20    | 2    | (2-1)/(7-1) | 0.1666       |
| 4    | 20    | 2    | (2-1)/(7-1) | 0.1666       |
| 5    | 30    | 5    | (5-1)/(7-1) | 0.6666       |
| 6    | 30    | 5    | (5-1)/(7-1) | 0.6666       |
| 7    | 40    | 7    | (7-1)/(7-1) | 1.0000       |

The return value range is 0 to 1, inclusive. The first row in any set has a PERCENT\_RANK of 0.

## Syntax

```
PERCENT_RANK ()
OVER (
[PARTITION BY partition_expression]
[ORDER BY order_list]
)
```

## Arguments

**()**

The function takes no arguments, but the empty parentheses are required.

**OVER**

A clause that specifies the window partitioning. The OVER clause cannot contain a window frame specification.

PARTITION BY *partition\_expression*

Optional. An expression that sets the range of records for each group in the OVER clause.

ORDER BY *order\_list*

Optional. The expression on which to calculate percent rank. The expression must have either a numeric data type or be implicitly convertible to one. If ORDER BY is omitted, the return value is 0 for all rows.

If ORDER BY does not produce a unique ordering, the order of the rows is nondeterministic. For more information, see [Unique ordering of data for window functions \(p. 1339\)](#).

## Return type

FLOAT8

## Examples

The following example calculates the percent rank of the sales quantities for each seller:

```
select sellerid, qty, percent_rank()
over (partition by sellerid order by qty)
from winsales;

sellerid qty percent_rank

1 10.00 0.0
1 10.64 0.5
1 30.37 1.0
3 10.04 0.0
3 15.15 0.33
3 20.75 0.67
3 30.55 1.0
2 20.09 0.0
2 20.12 1.0
4 10.12 0.0
4 40.23 1.0
```

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

## PERCENTILE\_CONT window function

PERCENTILE\_CONT is an inverse distribution function that assumes a continuous distribution model. It takes a percentile value and a sort specification, and returns an interpolated value that would fall into the given percentile value with respect to the sort specification.

PERCENTILE\_CONT computes a linear interpolation between values after ordering them. Using the percentile value (P) and the number of not null rows (N) in the aggregation group, the function computes the row number after ordering the rows according to the sort specification. This row number (RN) is computed according to the formula  $RN = (1 + (P * (N - 1)))$ . The final result of the aggregate function is computed by linear interpolation between the values from rows at row numbers CRN = CEILING(RN) and FRN = FLOOR(RN).

The final result will be as follows.

If (CRN = FRN = RN) then the result is (value of expression from row at RN)

Otherwise the result is as follows:

$(CRN - RN) * (\text{value of expression for row at FRN}) + (RN - FRN) * (\text{value of expression for row at CRN}).$

You can specify only the PARTITION clause in the OVER clause. If PARTITION is specified, for each row, PERCENTILE\_CONT returns the value that would fall into the specified percentile among a set of values within a given partition.

PERCENTILE\_CONT is a compute-node only function. The function returns an error if the query doesn't reference a user-defined table or Amazon Redshift system table.

## Syntax

```
PERCENTILE_CONT (percentile)
WITHIN GROUP (ORDER BY expr)
OVER ([PARTITION BY expr_list])
```

## Arguments

### *percentile*

Numeric constant between 0 and 1. Nulls are ignored in the calculation.

### WITHIN GROUP ( ORDER BY *expr* )

Specifies numeric or date/time values to sort and compute the percentile over.

### OVER

Specifies the window partitioning. The OVER clause cannot contain a window ordering or window frame specification.

### PARTITION BY *expr*

Optional argument that sets the range of records for each group in the OVER clause.

## Returns

The return type is determined by the data type of the ORDER BY expression in the WITHIN GROUP clause. The following table shows the return type for each ORDER BY expression data type.

| Input Type                         | Return Type |
|------------------------------------|-------------|
| INT2, INT4, INT8, NUMERIC, DECIMAL | DECIMAL     |
| FLOAT, DOUBLE                      | DOUBLE      |
| DATE                               | DATE        |
| TIMESTAMP                          | TIMESTAMP   |

## Usage notes

If the ORDER BY expression is a DECIMAL data type defined with the maximum precision of 38 digits, it is possible that PERCENTILE\_CONT will return either an inaccurate result or an error. If the return value of the PERCENTILE\_CONT function exceeds 38 digits, the result is truncated to fit, which causes a loss of precision. If, during interpolation, an intermediate result exceeds the maximum precision, a numeric overflow occurs and the function returns an error. To avoid these conditions, we recommend either using a data type with lower precision or casting the ORDER BY expression to a lower precision.

For example, a SUM function with a DECIMAL argument returns a default precision of 38 digits. The scale of the result is the same as the scale of the argument. So, for example, a SUM of a DECIMAL(5,2) column returns a DECIMAL(38,2) data type.

The following example uses a SUM function in the ORDER BY clause of a PERCENTILE\_CONT function. The data type of the PRICEPAID column is DECIMAL (8,2), so the SUM function returns DECIMAL(38,2).

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid) desc) over()
from sales where salesid < 10 group by salesid;
```

To avoid a potential loss of precision or an overflow error, cast the result to a DECIMAL data type with lower precision, as the following example shows.

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid)::decimal(30,2) desc) over()
from sales where salesid < 10 group by salesid;
```

## Examples

The following examples uses the WINSALES table. For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over() as median from winsales;

sellerid | qty | median
-----+-----+
 1 | 10 | 20.0
 1 | 10 | 20.0
 3 | 10 | 20.0
 4 | 10 | 20.0
 3 | 15 | 20.0
 2 | 20 | 20.0
 3 | 20 | 20.0
 2 | 20 | 20.0
 3 | 30 | 20.0
 1 | 30 | 20.0
 4 | 40 | 20.0
(11 rows)
```

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over(partition by sellerid) as median from winsales;

sellerid | qty | median
-----+-----+
 2 | 20 | 20.0
 2 | 20 | 20.0
 4 | 10 | 25.0
 4 | 40 | 25.0
 1 | 10 | 10.0
 1 | 10 | 10.0
 1 | 30 | 10.0
 3 | 10 | 17.5
 3 | 15 | 17.5
 3 | 20 | 17.5
 3 | 30 | 17.5
(11 rows)
```

The following example calculates the PERCENTILE\_CONT and PERCENTILE\_DISC of the ticket sales for sellers in Washington state.

```
SELECT sellerid, state, sum(qtysold*pricepaid) sales,
percentile_cont(0.6) within group (order by sum(qtysold*pricepaid)::decimal(14,2)) desc
over(),
percentile_disc(0.6) within group (order by sum(qtysold*pricepaid)::decimal(14,2)) desc
over()
from sales s, users u
where s.sellerid = u.userid and state = 'WA' and sellerid < 1000
group by sellerid, state;

sellerid | state | sales | percentile_cont | percentile_disc
-----+-----+-----+-----+-----+
127 | WA | 6076.00 | 2044.20 | 1531.00
787 | WA | 6035.00 | 2044.20 | 1531.00
381 | WA | 5881.00 | 2044.20 | 1531.00
777 | WA | 2814.00 | 2044.20 | 1531.00
33 | WA | 1531.00 | 2044.20 | 1531.00
800 | WA | 1476.00 | 2044.20 | 1531.00
1 | WA | 1177.00 | 2044.20 | 1531.00
(7 rows)
```

## PERCENTILE\_DISC window function

PERCENTILE\_DISC is an inverse distribution function that assumes a discrete distribution model. It takes a percentile value and a sort specification and returns an element from the given set.

For a given percentile value P, PERCENTILE\_DISC sorts the values of the expression in the ORDER BY clause and returns the value with the smallest cumulative distribution value (with respect to the same sort specification) that is greater than or equal to P.

You can specify only the PARTITION clause in the OVER clause.

PERCENTILE\_DISC is a compute-node only function. The function returns an error if the query doesn't reference a user-defined table or Amazon Redshift system table.

### Syntax

```
PERCENTILE_DISC (percentile)
WITHIN GROUP (ORDER BY expr)
OVER ([PARTITION BY expr_list])
```

### Arguments

#### *percentile*

Numeric constant between 0 and 1. Nulls are ignored in the calculation.

#### WITHIN GROUP ( ORDER BY *expr* )

Specifies numeric or date/time values to sort and compute the percentile over.

#### OVER

Specifies the window partitioning. The OVER clause cannot contain a window ordering or window frame specification.

#### PARTITION BY *expr*

Optional argument that sets the range of records for each group in the OVER clause.

## Returns

The same data type as the ORDER BY expression in the WITHIN GROUP clause.

## Examples

The following examples uses the WINSALES table. For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

```
select sellerid, qty, percentile_disc(0.5)
within group (order by qty)
over() as median from winsales;

sellerid | qty | median
-----+-----+
 1 | 10 | 20
 3 | 10 | 20
 1 | 10 | 20
 4 | 10 | 20
 3 | 15 | 20
 2 | 20 | 20
 2 | 20 | 20
 3 | 20 | 20
 1 | 30 | 20
 3 | 30 | 20
 4 | 40 | 20
(11 rows)
```

```
select sellerid, qty, percentile_disc(0.5)
within group (order by qty)
over(partition by sellerid) as median from winsales;

sellerid | qty | median
-----+-----+
 2 | 20 | 20
 2 | 20 | 20
 4 | 10 | 10
 4 | 40 | 10
 1 | 10 | 10
 1 | 10 | 10
 1 | 30 | 10
 3 | 10 | 15
 3 | 15 | 15
 3 | 20 | 15
 3 | 30 | 15
(11 rows)
```

## RANK window function

The RANK window function determines the rank of a value in a group of values, based on the ORDER BY expression in the OVER clause. If the optional PARTITION BY clause is present, the rankings are reset for each group of rows. Rows with equal values for the ranking criteria receive the same rank. Amazon Redshift adds the number of tied rows to the tied rank to calculate the next rank and thus the ranks might not be consecutive numbers. For example, if two rows are ranked 1, the next rank is 3.

RANK differs from the [DENSE\\_RANK window function \(p. 1345\)](#) in one respect: For DENSE\_RANK, if two or more rows tie, there is no gap in the sequence of ranked values. For example, if two rows are ranked 1, the next rank is 2.

You can have ranking functions with different PARTITION BY and ORDER BY clauses in the same query.

## Syntax

```
RANK () OVER
(
[PARTITION BY expr_list]
[ORDER BY order_list]
)
```

## Arguments

**()**

The function takes no arguments, but the empty parentheses are required.

**OVER**

The window clauses for the RANK function.

**PARTITION BY *expr\_list***

Optional. One or more expressions that define the window.

**ORDER BY *order\_list***

Optional. Defines the columns on which the ranking values are based. If no PARTITION BY is specified, ORDER BY uses the entire table. If ORDER BY is omitted, the return value is 1 for all rows.

If ORDER BY does not produce a unique ordering, the order of the rows is nondeterministic. For more information, see [Unique ordering of data for window functions \(p. 1339\)](#).

## Return type

INTEGER

## Examples

The following example orders the table by the quantity sold (default ascending), and assign a rank to each row. A rank value of 1 is the highest ranked value. The results are sorted after the window function results are applied:

```
select salesid, qty,
rank() over (order by qty) as rnk
from winsales
order by 2,1;

salesid | qty | rnk
-----+---+---
10001 | 10 | 1
10006 | 10 | 1
30001 | 10 | 1
40005 | 10 | 1
30003 | 15 | 5
20001 | 20 | 6
20002 | 20 | 6
30004 | 20 | 6
10005 | 30 | 9
30007 | 30 | 9
40001 | 40 | 11
(11 rows)
```

Note that the outer ORDER BY clause in this example includes columns 2 and 1 to make sure that Amazon Redshift returns consistently sorted results each time this query is run. For example, rows with

sales IDs 10001 and 10006 have identical QTY and RNK values. Ordering the final result set by column 1 ensures that row 10001 always falls before 10006. For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

In the following example, the ordering is reversed for the window function (`order by qty desc`). Now the highest rank value applies to the largest QTY value.

```
select salesid, qty,
rank() over (order by qty desc) as rank
from winsales
order by 2,1;

salesid | qty | rank
-----+---+---
10001 | 10 | 8
10006 | 10 | 8
30001 | 10 | 8
40005 | 10 | 8
30003 | 15 | 7
20001 | 20 | 4
20002 | 20 | 4
30004 | 20 | 4
10005 | 30 | 2
30007 | 30 | 2
40001 | 40 | 1
(11 rows)
```

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

The following example partitions the table by SELLERID and order each partition by the quantity (in descending order) and assign a rank to each row. The results are sorted after the window function results are applied.

```
select salesid, sellerid, qty, rank() over
(partition by sellerid
order by qty desc) as rank
from winsales
order by 2,3,1;

salesid | sellerid | qty | rank
-----+---+---+---
10001 | 1 | 10 | 2
10006 | 1 | 10 | 2
10005 | 1 | 30 | 1
20001 | 2 | 20 | 1
20002 | 2 | 20 | 1
30001 | 3 | 10 | 4
30003 | 3 | 15 | 3
30004 | 3 | 20 | 2
30007 | 3 | 30 | 1
40005 | 4 | 10 | 2
40001 | 4 | 40 | 1
(11 rows)
```

## RATIO\_TO\_REPORT window function

Calculates the ratio of a value to the sum of the values in a window or partition. The ratio to report value is determined using the formula:

`value of ratio_expression argument for the current row / sum of ratio_expression argument for the window or partition`

The following dataset illustrates use of this formula:

| Row# | Value | Calculation    | RATIO_TO_REPORT |
|------|-------|----------------|-----------------|
| 1    | 2500  | (2500)/(13900) | 0.1798          |
| 2    | 2600  | (2600)/(13900) | 0.1870          |
| 3    | 2800  | (2800)/(13900) | 0.2014          |
| 4    | 2900  | (2900)/(13900) | 0.2086          |
| 5    | 3100  | (3100)/(13900) | 0.2230          |

The return value range is 0 to 1, inclusive. If *ratio\_expression* is NULL, then the return value is NULL.

## Syntax

```
RATIO_TO_REPORT (ratio_expression)
OVER ([PARTITION BY partition_expression])
```

## Arguments

### *ratio\_expression*

An expression, such as a column name, that provides the value for which to determine the ratio. The expression must have either a numeric data type or be implicitly convertible to one.

You cannot use any other analytic function in *ratio\_expression*.

### OVER

A clause that specifies the window partitioning. The OVER clause cannot contain a window ordering or window frame specification.

### PARTITION BY *partition\_expression*

Optional. An expression that sets the range of records for each group in the OVER clause.

## Return type

FLOAT8

## Examples

The following example calculates the ratios of the sales quantities for each seller:

```
select sellerid, qty, ratio_to_report(qty)
over (partition by sellerid)
from winsales;

sellerid qty ratio_to_report

2 20.12312341 0.5
2 20.08630000 0.5
4 10.12414400 0.2
4 40.23000000 0.8
1 30.37262000 0.6
1 10.64000000 0.21
1 10.00000000 0.2
3 10.03500000 0.13
3 15.14660000 0.2
3 30.54790000 0.4
3 20.74630000 0.27
```

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

## ROW\_NUMBER window function

Determines the ordinal number of the current row within a group of rows, counting from 1, based on the ORDER BY expression in the OVER clause. If the optional PARTITION BY clause is present, the ordinal numbers are reset for each group of rows. Rows with equal values for the ORDER BY expressions receive the different row numbers nondeterministically.

### Syntax

```
ROW_NUMBER () OVER
(
[PARTITION BY expr_list]
[ORDER BY order_list]
)
```

### Arguments

()

The function takes no arguments, but the empty parentheses are required.

OVER

The window clauses for the ROW\_NUMBER function.

PARTITION BY *expr\_list*

Optional. One or more expressions that define the ROW\_NUMBER function.

ORDER BY *order\_list*

Optional. The expression that defines the columns on which the row numbers are based. If no PARTITION BY is specified, ORDER BY uses the entire table.

If ORDER BY does not produce a unique ordering or is omitted, the order of the rows is nondeterministic. For more information, see [Unique ordering of data for window functions \(p. 1339\)](#).

### Return type

BIGINT

### Examples

The following example partitions the table by SELLERID and orders each partition by QTY (in ascending order), then assigns a row number to each row. The results are sorted after the window function results are applied.

```
select salesid, sellerid, qty,
row_number() over
(partition by sellerid
 order by qty asc) as row
from winsales
order by 2,4;

salesid | sellerid | qty | row
-----+-----+-----+
```

|           |   |    |   |
|-----------|---|----|---|
| 10006     | 1 | 10 | 1 |
| 10001     | 1 | 10 | 2 |
| 10005     | 1 | 30 | 3 |
| 20001     | 2 | 20 | 1 |
| 20002     | 2 | 20 | 2 |
| 30001     | 3 | 10 | 1 |
| 30003     | 3 | 15 | 2 |
| 30004     | 3 | 20 | 3 |
| 30007     | 3 | 30 | 4 |
| 40005     | 4 | 10 | 1 |
| 40001     | 4 | 40 | 2 |
| (11 rows) |   |    |   |

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

## STDDEV\_SAMP and STDDEV\_POP window functions

The STDDEV\_SAMP and STDDEV\_POP window functions return the sample and population standard deviation of a set of numeric values (integer, decimal, or floating-point). See also [STDDEV\\_SAMP and STDDEV\\_POP functions \(p. 1037\)](#).

STDDEV\_SAMP and STDDEV are synonyms for the same function.

### Syntax

```
STDDEV_SAMP | STDDEV | STDDEV_POP
([ALL] expression) OVER
(
[PARTITION BY expr_list]
[ORDER BY order_list
 frame_clause]
)
```

### Arguments

#### *expression*

The target column or expression that the function operates on.

#### ALL

With the argument ALL, the function retains all duplicate values from the expression. ALL is the default. DISTINCT is not supported.

#### OVER

Specifies the window clauses for the aggregation functions. The OVER clause distinguishes window aggregation functions from normal set aggregation functions.

#### PARTITION BY *expr\_list*

Defines the window for the function in terms of one or more expressions.

#### ORDER BY *order\_list*

Sorts the rows within each partition. If no PARTITION BY is specified, ORDER BY uses the entire table.

#### *frame\_clause*

If an ORDER BY clause is used for an aggregate function, an explicit frame clause is required. The frame clause refines the set of rows in a function's window, including or excluding sets of rows within

the ordered result. The frame clause consists of the ROWS keyword and associated specifiers. See [Window function syntax summary \(p. 1336\)](#).

## Data types

The argument types supported by the STDDEV functions are SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL, and DOUBLE PRECISION.

Regardless of the data type of the expression, the return type of a STDDEV function is a double precision number.

## Examples

The following example shows how to use STDDEV\_POP and VAR\_POP functions as window functions. The query computes the population variance and population standard deviation for PRICEPAID values in the SALES table.

```
select salesid, dateid, pricepaid,
round(stddev_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as stddevpop,
round(var_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as varpop
from sales
order by 2,1;

salesid | dateid | pricepaid | stddevpop | varpop
-----+-----+-----+-----+
 33095 | 1827 | 234.00 | 0 | 0
 65082 | 1827 | 472.00 | 119 | 14161
 88268 | 1827 | 836.00 | 248 | 61283
 97197 | 1827 | 708.00 | 230 | 53019
110328 | 1827 | 347.00 | 223 | 49845
110917 | 1827 | 337.00 | 215 | 46159
150314 | 1827 | 688.00 | 211 | 44414
157751 | 1827 | 1730.00 | 447 | 199679
165890 | 1827 | 4192.00 | 1185 | 1403323
...
```

The sample standard deviation and variance functions can be used in the same way.

## SUM window function

The SUM window function returns the sum of the input column or expression values. The SUM function works with numeric values and ignores NULL values.

### Syntax

```
SUM ([ALL] expression) OVER
(
[PARTITION BY expr_list]
[ORDER BY order_list
 frame_clause]
)
```

### Arguments

*expression*

The target column or expression that the function operates on.

## ALL

With the argument ALL, the function retains all duplicate values from the expression. ALL is the default. DISTINCT is not supported.

## OVER

Specifies the window clauses for the aggregation functions. The OVER clause distinguishes window aggregation functions from normal set aggregation functions.

### PARTITION BY *expr\_list*

Defines the window for the SUM function in terms of one or more expressions.

### ORDER BY *order\_list*

Sorts the rows within each partition. If no PARTITION BY is specified, ORDER BY uses the entire table.

### *frame\_clause*

If an ORDER BY clause is used for an aggregate function, an explicit frame clause is required. The frame clause refines the set of rows in a function's window, including or excluding sets of rows within the ordered result. The frame clause consists of the ROWS keyword and associated specifiers. See [Window function syntax summary \(p. 1336\)](#).

## Data types

The argument types supported by the SUM function are SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL, and DOUBLE PRECISION.

The return types supported by the SUM function are:

- BIGINT for SMALLINT or INTEGER arguments
- NUMERIC for BIGINT arguments
- DOUBLE PRECISION for floating-point arguments

## Examples

The following example creates a cumulative (rolling) sum of sales quantities ordered by date and sales ID:

```
select salesid, dateid, sellerid, qty,
sum(qty) over (order by dateid, salesid rows unbounded preceding) as sum
from winsales
order by 2,1;

salesid | dateid | sellerid | qty | sum
-----+-----+-----+-----+
30001 | 2003-08-02 | 3 | 10 | 10
10001 | 2003-12-24 | 1 | 10 | 20
10005 | 2003-12-24 | 1 | 30 | 50
40001 | 2004-01-09 | 4 | 40 | 90
10006 | 2004-01-18 | 1 | 10 | 100
20001 | 2004-02-12 | 2 | 20 | 120
40005 | 2004-02-12 | 4 | 10 | 130
20002 | 2004-02-16 | 2 | 20 | 150
30003 | 2004-04-18 | 3 | 15 | 165
30004 | 2004-04-18 | 3 | 20 | 185
30007 | 2004-09-07 | 3 | 30 | 215
(11 rows)
```

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

The following example creates a cumulative (rolling) sum of sales quantities by date, partition the results by seller ID, and order the results by date and sales ID within the partition:

```
select salesid, dateid, sellerid, qty,
sum(qty) over (partition by sellerid
order by dateid, salesid rows unbounded preceding) as sum
from winsales
order by 2,1;

salesid | dateid | sellerid | qty | sum
-----+-----+-----+-----+
30001 | 2003-08-02 | | 3 | 10
10001 | 2003-12-24 | | 1 | 10
10005 | 2003-12-24 | | 1 | 30
40001 | 2004-01-09 | | 4 | 40
10006 | 2004-01-18 | | 1 | 50
20001 | 2004-02-12 | | 2 | 20
40005 | 2004-02-12 | | 4 | 50
20002 | 2004-02-16 | | 2 | 20
30003 | 2004-04-18 | | 3 | 15
30004 | 2004-04-18 | | 3 | 20
30007 | 2004-09-07 | | 3 | 75
(11 rows)
```

The following example numbers all of the rows sequentially in the result set, ordered by the SELLERID and SALESID columns:

```
select salesid, sellerid, qty,
sum(1) over (order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

salesid | sellerid | qty | rownum
-----+-----+-----+
10001 | 1 | 10 | 1
10005 | 1 | 30 | 2
10006 | 1 | 10 | 3
20001 | 2 | 20 | 4
20002 | 2 | 20 | 5
30001 | 3 | 10 | 6
30003 | 3 | 15 | 7
30004 | 3 | 20 | 8
30007 | 3 | 30 | 9
40001 | 4 | 40 | 10
40005 | 4 | 10 | 11
(11 rows)
```

For a description of the WINSALES table, see [Sample table for window function examples \(p. 1340\)](#).

The following example numbers all rows sequentially in the result set, partition the results by SELLERID, and order the results by SELLERID and SALESID within the partition:

```
select salesid, sellerid, qty,
sum(1) over (partition by sellerid
order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

salesid | sellerid | qty | rownum
-----+-----+-----+
```

|           |   |    |   |
|-----------|---|----|---|
| 10001     | 1 | 10 | 1 |
| 10005     | 1 | 30 | 2 |
| 10006     | 1 | 10 | 3 |
| 20001     | 2 | 20 | 1 |
| 20002     | 2 | 20 | 2 |
| 30001     | 3 | 10 | 1 |
| 30003     | 3 | 15 | 2 |
| 30004     | 3 | 20 | 3 |
| 30007     | 3 | 30 | 4 |
| 40001     | 4 | 40 | 1 |
| 40005     | 4 | 10 | 2 |
| (11 rows) |   |    |   |

## VAR\_SAMP and VAR\_POP window functions

The VAR\_SAMP and VAR\_POP window functions return the sample and population variance of a set of numeric values (integer, decimal, or floating-point). See also [VAR\\_SAMP and VAR\\_POP functions \(p. 1040\)](#).

VAR\_SAMP and VARIANCE are synonyms for the same function.

### Syntax

```
VAR_SAMP | VARIANCE | VAR_POP
([ALL] expression) OVER
(
[PARTITION BY expr_list]
[ORDER BY order_list
 frame_clause]
)
```

### Arguments

#### *expression*

The target column or expression that the function operates on.

#### ALL

With the argument ALL, the function retains all duplicate values from the expression. ALL is the default. DISTINCT is not supported.

#### OVER

Specifies the window clauses for the aggregation functions. The OVER clause distinguishes window aggregation functions from normal set aggregation functions.

#### PARTITION BY *expr\_list*

Defines the window for the function in terms of one or more expressions.

#### ORDER BY *order\_list*

Sorts the rows within each partition. If no PARTITION BY is specified, ORDER BY uses the entire table.

#### *frame\_clause*

If an ORDER BY clause is used for an aggregate function, an explicit frame clause is required. The frame clause refines the set of rows in a function's window, including or excluding sets of rows within the ordered result. The frame clause consists of the ROWS keyword and associated specifiers. See [Window function syntax summary \(p. 1336\)](#).

## Data types

The argument types supported by the VARIANCE functions are SMALLINT, INTEGER, BIGINT, NUMERIC, DECIMAL, REAL, and DOUBLE PRECISION.

Regardless of the data type of the expression, the return type of a VARIANCE function is a double precision number.

# System administration functions

## Topics

- [CHANGE\\_QUERY\\_PRIORITY \(p. 1379\)](#)
- [CHANGE\\_SESSION\\_PRIORITY \(p. 1380\)](#)
- [CHANGE\\_USER\\_PRIORITY \(p. 1381\)](#)
- [CURRENT\\_SETTING \(p. 1382\)](#)
- [PG\\_CANCEL\\_BACKEND \(p. 1383\)](#)
- [PG\\_TERMINATE\\_BACKEND \(p. 1384\)](#)
- [REBOOT\\_CLUSTER \(p. 1385\)](#)
- [SET\\_CONFIG \(p. 1385\)](#)

Amazon Redshift supports several system administration functions.

## CHANGE\_QUERY\_PRIORITY

CHANGE\_QUERY\_PRIORITY enables superusers to modify the priority of a query that is either running or waiting in workload management (WLM).

This function enables superusers to immediately change the priority of any query in the system. Only one query, user, or session can run with the priority CRITICAL.

### Syntax

```
CHANGE_QUERY_PRIORITY(query_id, priority)
```

### Arguments

*query\_id*

The query identifier of the query whose priority is changed.

*priority*

The new priority to be assigned to the query. This argument must be a string with the value CRITICAL, HIGHEST, HIGH, NORMAL, LOW, or LOWEST.

### Return Type

None

### Example

The following example shows the column `query_priority` in the `STV_WLM_QUERY_STATE` system table.

```
select query, service_class, query_priority, state
from stv_wlm_query_state where service_class = 101;
query | service_class | query_priority | state
-----+-----+-----+-----
1076 | 101 | Lowest | Running
1075 | 101 | Lowest | Running
(2 rows)
```

The following example shows the results of a superuser running the function `change_query_priority` to change the priority to CRITICAL.

```
select change_query_priority(1076, 'Critical');

change_query_priority

Succeeded to change query priority. Priority changed from Lowest to Critical.
(1 row)
```

## CHANGE\_SESSION\_PRIORITY

`CHANGE_SESSION_PRIORITY` enables superusers to immediately change the priority of any session in the system. Only one session, user, or query can run with the priority CRITICAL.

### Syntax

```
CHANGE_SESSION_PRIORITY(pid, priority)
```

### Arguments

*pid*

The process identifier of the session whose priority is changed. The value -1 refers to the current session.

*priority*

The new priority to be assigned to the session. This argument must be a string with the value CRITICAL, HIGHEST, HIGH, NORMAL, LOW, or LOWEST.

### Return type

None

### Example

The following example returns the process identifier of the server process handling the current session.

```
select pg_backend_pid();

pg_backend_pid

30311
(1 row)
```

In this example, the priority is changed for the current session to LOWEST.

```
select change_session_priority(30311, 'Lowest');
```

```
change_session_priority

Succeeded to change session priority. Changed session (pid:30311) priority to lowest.
(1 row)
```

In this example, the priority is changed for the current session to HIGH.

```
select change_session_priority(-1, 'High');
change_session_priority

Succeeded to change session priority. Changed session (pid:30311) priority from lowest to
high.
(1 row)
```

In the following example, a stored procedure is created to change a session priority. Permission to run this stored procedure is granted to the database user test\_user.

```
CREATE OR REPLACE PROCEDURE sp_priority_low(pid IN int, result OUT varchar)
AS $$$
BEGIN
 select change_session_priority(pid, 'low') into result;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;
GRANT EXECUTE ON PROCEDURE sp_priority_low(int) TO test_user;
```

Then the database user named test\_user calls the procedure.

```
call sp_priority_low(pg_backend_pid());
result

Success. Change session (pid:13155) priority to low.
```

## CHANGE\_USER\_PRIORITY

CHANGE\_USER\_PRIORITY enables superusers to modify the priority of all queries issued by a user that are either running or waiting in workload management (WLM). Only one user, session, or query can run with the priority CRITICAL.

### Syntax

```
CHANGE_USER_PRIORITY(user_name, priority)
```

### Arguments

*user\_name*

The database user name whose query priority is changed.

*priority*

The new priority to be assigned to all queries issued by *user\_name*. This argument must be a string with the value CRITICAL, HIGHEST, HIGH, NORMAL, LOW, LOWEST, or RESET. Only superusers can change the priority to CRITICAL. Changing the priority to RESET removes the priority setting for *user\_name*.

## Return type

None

## Example

In the following example, the priority is changed for the user analysis\_user to LOWEST.

```
select change_user_priority('analysis_user', 'lowest');
 change_user_priority

Succeeded to change user priority. Changed user (analysis_user) priority to lowest.
(1 row)
```

In the next statement, the priority is changed to LOW.

```
select change_user_priority('analysis_user', 'low');
 change_user_priority

Succeeded to change user priority. Changed user (analysis_user) priority from Lowest to
low.
(1 row)
```

In this example, the priority is reset.

```
select change_user_priority('analysis_user', 'reset');
 change_user_priority

Succeeded to reset priority for user (analysis_user).
(1 row)
```

## CURRENT\_SETTING

CURRENT\_SETTING returns the current value of the specified configuration parameter.

This function is equivalent to the [SHOW \(p. 981\)](#) command.

### Syntax

```
current_setting('parameter')
```

The following statement returns the current value of the specified session context variable.

```
current_setting('variable_name')
current_setting('variable_name', error_if_undefined)
```

### Argument

*parameter*

Parameter value to display. For a list of configuration parameters, see [Configuration reference \(p. 1685\)](#)

*variable\_name*

The name of the variable to display. This must be a string constant for session context variables.

#### *error\_if\_undefined*

An optional boolean value that specifies the behavior if the variable name doesn't exist. When `error_if_undefined` is set to TRUE, which is the default, Amazon Redshift throws an error. When `error_if_undefined` is set to FALSE, Amazon Redshift returns null. Amazon Redshift supports the `error_if_undefined` parameter only for session context variables. This can't be used when the input is a configuration parameter.

#### Return type

Returns a CHAR or VARCHAR string.

#### Example

The following query returns the current setting for the `query_group` parameter:

```
select current_setting('query_group');

current_setting

unset
(1 row)
```

The following query returns the current setting for the variable.

```
SELECT current_setting('app_context.user_id', FALSE);
```

## PG\_CANCEL\_BACKEND

Cancels a query. `PG_CANCEL_BACKEND` is functionally equivalent to the [CANCEL \(p. 672\)](#) command. You can cancel queries currently being run by your user. Superusers can cancel any query.

#### Syntax

```
pg_cancel_backend(pid)
```

#### Arguments

*pid*

The process ID (PID) of the query to be canceled. You cannot cancel a query by specifying a query ID; you must specify the query's process ID. Requires an integer value.

#### Return type

None

#### Usage Notes

If queries in multiple sessions hold locks on the same table, you can use the [PG\\_TERMINATE\\_BACKEND \(p. 1384\)](#) function to terminate one of the sessions, which forces any currently running transactions in the terminated session to release all locks and roll back the transaction. Query the PG\_LOCKS catalog table to view currently held locks. If you cannot cancel a query because

if it is in transaction block (BEGIN ... END), you can terminate the session in which the query is running by using the PG\_TERMINATE\_BACKEND function.

## Examples

To cancel a currently running query, first retrieve the process ID for the query that you want to cancel. To determine the process IDs for all currently running queries, run the following command:

```
select pid, trim(starttime) as start,
duration, trim(user_name) as user,
substring (query,1,40) as querytxt
from stv_recents
where status = 'Running';

pid | starttime | duration | user | querytxt
-----+-----+-----+-----+-----+
 802 | 2013-10-14 09:19:03.55 | 132 | dwuser | select venuename from venue
 834 | 2013-10-14 08:33:49.47 | 1250414 | dwuser | select * from listing;
 964 | 2013-10-14 08:30:43.29 | 326179 | dwuser | select sellerid from sales
```

The following statement cancels the query with process ID 802:

```
select pg_cancel_backend(802);
```

## PG\_TERMINATE\_BACKEND

Terminates a session. You can terminate a session owned by your user. A superuser can terminate any session.

### Syntax

```
pg_terminate_backend(pid)
```

### Arguments

*pid*

The process ID of the session to be terminated. Requires an integer value.

### Return type

None

### Usage notes

If you are close to reaching the limit for concurrent connections, use PG\_TERMINATE\_BACKEND to terminate idle sessions and free up the connections. For more information, see [Limits in Amazon Redshift](#).

If queries in multiple sessions hold locks on the same table, you can use PG\_TERMINATE\_BACKEND to terminate one of the sessions, which forces any currently running transactions in the terminated session to release all locks and roll back the transaction. Query the PG\_LOCKS catalog table to view currently held locks.

If a query is not in a transaction block (BEGIN ... END), you can cancel the query by using the [CANCEL \(p. 672\)](#) command or the [PG\\_CANCEL\\_BACKEND \(p. 1383\)](#) function.

## Examples

The following statement queries the SVV\_TRANSACTIONS table to view all locks in effect for current transactions:

```
select * from svv_transactions;
txnid | lock_mode | relation | granted
-----+-----+-----+-----+
 | | | |
rsuser | dev | 96178 | 8585 | 2017-04-12 20:13:07 | AccessShareLock | relation
 | 51940 | true | | | |
rsuser | dev | 96178 | 8585 | 2017-04-12 20:13:07 | AccessShareLock | relation
 | 52000 | true | | | |
rsuser | dev | 96178 | 8585 | 2017-04-12 20:13:07 | AccessShareLock | relation
 | 108623 | true | | | |
rsuser | dev | 96178 | 8585 | 2017-04-12 20:13:07 | ExclusiveLock | transactionid
 | | | | | true
```

The following statement terminates the session holding the locks:

```
select pg_terminate_backend(8585);
```

## REBOOT\_CLUSTER

Reboot the Amazon Redshift cluster without closing the connections to the cluster. You must be a database superuser to run this command.

After this soft reboot has completed, the Amazon Redshift cluster returns an error to the user application and requires the user application to resubmit any transactions or queries interrupted by the soft reboot.

### Syntax

```
select reboot_cluster();
```

## SET\_CONFIG

Sets a configuration parameter to a new setting.

This function is equivalent to the SET command in SQL.

### Syntax

```
set_config('parameter', 'new_value' , is_local)
```

The following statement sets a session context variable to a new setting.

```
set_config('variable_name', 'new_value' , is_local)
```

### Arguments

*parameter*

Parameter to set.

*variable\_name*

The name of the variable to set.

*new\_value*

New value of the parameter.

*is\_local*

If true, parameter value applies only to the current transaction. Valid values are true or 1 and false or 0.

## Return type

Returns a CHAR or VARCHAR string.

## Examples

The following query sets the value of the `query_group` parameter to test for the current transaction only:

```
select set_config('query_group', 'test', true);
set_config

test
(1 row)
```

The following example sets session context variables.

```
SELECT set_config('app.username', 'cuddy', FALSE);
```

# System information functions

Amazon Redshift supports numerous system information functions.

## Topics

- [CURRENT\\_AWS\\_ACCOUNT \(p. 1387\)](#)
- [CURRENT\\_DATABASE \(p. 1387\)](#)
- [CURRENT\\_NAMESPACE \(p. 1388\)](#)
- [CURRENT\\_SCHEMA \(p. 1388\)](#)
- [CURRENT\\_SCHEMAS \(p. 1389\)](#)
- [CURRENT\\_USER \(p. 1389\)](#)
- [CURRENT\\_USER\\_ID \(p. 1390\)](#)
- [DEFAULT\\_IAM\\_ROLE \(p. 1390\)](#)
- [HAS\\_ASSUMEROLE\\_PRIVILEGE \(p. 1391\)](#)
- [HAS\\_DATABASE\\_PRIVILEGE \(p. 1392\)](#)
- [HAS\\_SCHEMA\\_PRIVILEGE \(p. 1392\)](#)
- [HAS\\_TABLE\\_PRIVILEGE \(p. 1393\)](#)
- [PG\\_BACKEND\\_PID \(p. 1394\)](#)
- [PG\\_GET\\_COLS \(p. 1395\)](#)
- [PG\\_GET\\_GRANTEE\\_BY\\_IAM\\_ROLE \(p. 1396\)](#)
- [PG\\_GET\\_IAM\\_ROLE\\_BY\\_USER \(p. 1397\)](#)

- [PG\\_GET\\_LATE\\_BINDING\\_VIEW\\_COLS \(p. 1398\)](#)
- [PG\\_LAST\\_COPY\\_COUNT \(p. 1399\)](#)
- [PG\\_LAST\\_COPY\\_ID \(p. 1400\)](#)
- [PG\\_LAST\\_UNLOAD\\_ID \(p. 1401\)](#)
- [PG\\_LAST\\_QUERY\\_ID \(p. 1401\)](#)
- [PG\\_LAST\\_UNLOAD\\_COUNT \(p. 1402\)](#)
- [SLICE\\_NUM Function \(p. 1403\)](#)
- [USER \(p. 1403\)](#)
- [ROLE\\_IS\\_MEMBER\\_OF \(p. 1403\)](#)
- [USER\\_IS\\_MEMBER\\_OF \(p. 1404\)](#)
- [VERSION \(p. 1405\)](#)

## CURRENT\_AWS\_ACCOUNT

Returns the AWS account associated with the Amazon Redshift cluster that submitted a query.

### Syntax

```
current_aws_account
```

### Return type

Returns an integer.

### Example

The following query returns the name of the current database.

```
select user, current_aws_account;
current_user | current_account
-----+-----
dwuser | 987654321
(1 row)
```

## CURRENT\_DATABASE

Returns the name of the database where you are currently connected.

### Syntax

```
current_database()
```

### Return type

Returns a CHAR or VARCHAR string.

### Example

The following query returns the name of the current database.

```
select current_database();
```

```
current_database

tickit
(1 row)
```

## CURRENT\_NAMESPACE

Returns the cluster namespace of the current Amazon Redshift cluster. Amazon Redshift cluster namespace is the unique ID of the Amazon Redshift cluster.

### Syntax

```
current_namespace
```

### Return type

Returns a CHAR or VARCHAR string.

### Example

The following query returns the name of the current namespace.

```
select user, current_namespace;
current_user | current_namespace

dwuser | 86b5169f-01dc-4a6f-9fb8-e2e24359e9a8
(1 row)
```

## CURRENT\_SCHEMA

Returns the name of the schema at the front of the search path. This schema will be used for any tables or other named objects that are created without specifying a target schema.

### Syntax

#### Note

This is a leader-node function. This function returns an error if it references a user-created table, an STL or STV system table, or an SVV or SVL system view.

```
current_schema()
```

### Return type

CURRENT\_SCHEMA returns a CHAR or VARCHAR string.

### Examples

The following query returns the current schema:

```
select current_schema();
current_schema

public
```

(1 row)

## CURRENT\_SCHEMAS

Returns an array of the names of any schemas in the current search path. The current search path is defined in the search\_path parameter.

### Syntax

#### Note

This is a leader-node function. This function returns an error if it references a user-created table, an STL or STV system table, or an SVV or SVL system view.

```
current_schemas(include_implicit)
```

### Argument

*include\_implicit*

If true, specifies that the search path should include any implicitly included system schemas. Valid values are true and false. Typically, if true, this parameter returns the pg\_catalog schema in addition to the current schema.

### Return type

Returns a CHAR or VARCHAR string.

### Examples

The following example returns the names of the schemas in the current search path, not including implicitly included system schemas:

```
select current_schemas(false);

current_schemas

{public}
(1 row)
```

The following example returns the names of the schemas in the current search path, including implicitly included system schemas:

```
select current_schemas(true);

current_schemas

{pg_catalog,public}
(1 row)
```

## CURRENT\_USER

Returns the user name of the current "effective" user of the database, as applicable to checking permissions. Usually, this user name will be the same as the session user; however, this can occasionally be changed by superusers.

#### Note

Do not use trailing parentheses when calling CURRENT\_USER.

## Syntax

```
current_user
```

## Return type

CURRENT\_USER returns a CHAR or VARCHAR string.

## Example

The following query returns the name of the current database user:

```
select current_user;

current_user

dwuser
(1 row)
```

## CURRENT\_USER\_ID

Returns the unique identifier for the Amazon Redshift user logged in to the current session.

## Syntax

```
CURRENT_USER_ID
```

## Return type

The CURRENT\_USER\_ID function returns an integer.

## Examples

The following example returns the user name and current user ID for this session:

```
select user, current_user_id;

current_user | current_user_id
-----+-----
dwuser | 1
(1 row)
```

## DEFAULT\_IAM\_ROLE

Returns the default IAM role currently associated with the Amazon Redshift cluster. The function returns none if there isn't any default IAM role associated.

## Syntax

```
select default_iam_role();
```

## Return type

Returns a VARCHAR string.

## Example

The following example returns the default IAM role currently associated with the specified Amazon Redshift cluster,

```
select default_iam_role();
 default_iam_role

arn:aws:iam::123456789012:role/myRedshiftRole
(1 row)
```

## HAS\_ASSUMEROLE\_PRIVILEGE

Returns Boolean true (t) if the specified user has the specified IAM role with the privilege to run the specified command. The function returns false (f) if the user doesn't have the specified IAM role with the privilege to run the specified command. For more information about privileges, see [GRANT \(p. 887\)](#).

### Syntax

```
has_assumerole_privilege([user,] iam_role_arn, cmd_type)
```

### Arguments

*user*

The name of the user to check for IAM role privileges. The default is to check the current user. Superusers and users can use this function. However, users can only view their own privileges.

*iam\_role\_arn*

The IAM role that has been granted the command privileges.

*cmd\_type*

The command for which access has been granted. Valid values are the following:

- COPY
- UNLOAD
- EXTERNAL FUNCTION
- CREATE MODEL

### Return type

BOOLEAN

## Example

The following query confirms that the user `reg_user1` has the privilege for the Redshift-S3-Read role to run the COPY command.

```
select has_assumerole_privilege('reg_user1', 'arn:aws:iam::123456789012:role/Redshift-S3-
Read', 'copy');
```

```
has_assumerole_privilege

true
(1 row)
```

## HAS\_DATABASE\_PRIVILEGE

Returns `true` if the user has the specified privilege for the specified database. For more information about privileges, see [GRANT \(p. 887\)](#).

### Syntax

#### Note

This is a leader-node function. This function returns an error if it references a user-created table, an STL or STV system table, or an SVV or SVL system view.

```
has_database_privilege([user,] database, privilege)
```

### Arguments

*user*

The name of the user to check for database privileges. The default is to check the current user.

*database*

The database associated with the privilege.

*privilege*

The privilege to check. Valid values are the following:

- CREATE
- TEMPORARY
- TEMP

### Return type

Returns a CHAR or VARCHAR string.

### Example

The following query confirms that the GUEST user has the TEMP privilege on the TICKIT database.

```
select has_database_privilege('guest', 'tickit', 'temp');

has_database_privilege

true
(1 row)
```

## HAS\_SCHEMA\_PRIVILEGE

Returns `true` if the user has the specified privilege for the specified schema. For more information about privileges, see [GRANT \(p. 887\)](#).

### Syntax

#### Note

This is a leader-node function. This function returns an error if it references a user-created table, an STL or STV system table, or an SVV or SVL system view.

```
has_schema_privilege([user,] schema, privilege)
```

## Arguments

*user*

The name of the user to check for schema privileges. The default is to check the current user.  
*schema*

The schema associated with the privilege.  
*privilege*

The privilege to check. Valid values are the following:

- CREATE
- USAGE

## Return type

Returns a CHAR or VARCHAR string.

## Example

The following query confirms that the GUEST user has the CREATE privilege on the PUBLIC schema:

```
select has_schema_privilege('guest', 'public', 'create');

has_schema_privilege

true
(1 row)
```

## HAS\_TABLE\_PRIVILEGE

Returns true if the user has the specified privilege for the specified table.

## Syntax

### Note

This is a leader-node function. This function returns an error if it references a user-created table, an STL or STV system table, or an SVV or SVL system view. For more information about privileges, see [GRANT \(p. 887\)](#).

```
has_table_privilege([user,] table, privilege)
```

## Arguments

*user*

The name of the user to check for table privileges. The default is to check the current user.  
*table*

Table associated with the privilege.  
*privilege*

Privilege to check. Valid values are the following:

- SELECT
- INSERT

- UPDATE
- DELETE
- REFERENCES

## Return type

Returns a CHAR or VARCHAR string.

## Examples

The following query finds that the GUEST user doesn't have SELECT privilege on the LISTING table.

```
select has_table_privilege('guest', 'listing', 'select');

has_table_privilege

false
(1 row)
```

The following query lists table privileges, including select, insert, update, and delete, using output from the pg\_tables and pg\_user catalog tables. This is a sample only. You might have to specify a schema name and table names from your database. For more information, see [Querying the catalog tables \(p. 1680\)](#).

```
SELECT
 tablename
 ,username
 ,HAS_TABLE_PRIVILEGE(users.username, tablename, 'select') AS sel
 ,HAS_TABLE_PRIVILEGE(users.username, tablename, 'insert') AS ins
 ,HAS_TABLE_PRIVILEGE(users.username, tablename, 'update') AS upd
 ,HAS_TABLE_PRIVILEGE(users.username, tablename, 'delete') AS del
FROM
(SELECT * from pg_tables
WHERE schemaname = 'public' and tablename in ('event','listing')) as tables
,(SELECT * FROM pg_user) AS users;

tablename | username | sel | ins | upd | del

event john true true true true
event sally false false false false
event elsa false false false false
listing john true true true true
listing sally false false false false
listing elsa false false false false
```

In the sample, the schema name in the WHERE clause limits returned records to tables in the *public* schema. The query also contains a cross join. For more information, see [JOIN examples \(p. 946\)](#).

## PG\_BACKEND\_PID

Returns the process ID (PID) of the server process handling the current session.

### Note

The PID is not globally unique. It can be reused over time.

## Syntax

```
pg_backend_pid()
```

## Return type

Returns an integer.

## Example

You can correlate PG\_BACKEND\_PID with log tables to retrieve information for the current session. For example, the following query returns the query ID and a portion of the query text for queries completed in the current session.

```
select query, substring(text,1,40)
from stl_querytext
where pid = PG_BACKEND_PID()
order by query desc;

query | substring
-----+-----
14831 | select query, substring(text,1,40) from
14827 | select query, substring(path,0,80) as pa
14826 | copy category from 's3://dw-tickit/manif
14825 | Count rows in target table
14824 | unload ('select * from category') to 's3
(5 rows)
```

You can correlate PG\_BACKEND\_PID with the pid column in the following log tables (exceptions are noted in parentheses):

- [STL\\_CONNECTION\\_LOG \(p. 1505\)](#)
- [STL\\_DDLTEXT \(p. 1507\)](#)
- [STL\\_ERROR \(p. 1513\)](#)
- [STL\\_QUERY \(p. 1539\)](#)
- [STL\\_QUERYTEXT \(p. 1544\)](#)
- [STL\\_SESSIONS \(p. 1557\) \(process\)](#)
- [STL\\_TR\\_CONFLICT \(p. 1561\)](#)
- [STL.UtilityText \(p. 1567\)](#)
- [STV\\_ACTIVE\\_CURSORS \(p. 1578\)](#)
- [STV\\_INFLIGHT \(p. 1583\)](#)
- [STV\\_LOCKS \(p. 1586\) \(lock\\_owner\\_pid\)](#)
- [STV\\_RECENTS \(p. 1596\) \(process\\_id\)](#)

## PG\_GET\_COLS

Returns the column metadata for a table or view definition.

### Syntax

```
pg_get_cols('name')
```

### Arguments

*name*

The name of an Amazon Redshift table or view.

## Return type

VARCHAR

## Usage notes

The PG\_GET\_COLS function returns one row for each column in the table or view definition. The row contains a comma-separated list with the schema name, relation name, column name, data type, and column number.

## Example

The following example returns the column metadata for a view named SALES\_VW.

```
select pg_get_cols('sales_vw');

pg_get_cols

(public,sales_vw,salesid,integer,1)
(public,sales_vw,listid,integer,2)
(public,sales_vw,sellerid,integer,3)
(public,sales_vw,buyerid,integer,4)
(public,sales_vw,eventid,integer,5)
(public,sales_vw,dateid,smallint,6)
(public,sales_vw,qtysold,smallint,7)
(public,sales_vw,pricepaid,"numeric(8,2)",8)
(public,sales_vw,commission,"numeric(8,2)",9)
(public,sales_vw,saletime,"timestamp without time zone",10)
```

The following example returns the column metadata for the SALES\_VW view in table format.

| view_schema | view_name | col_name   | col_type                    | col_num |
|-------------|-----------|------------|-----------------------------|---------|
| public      | sales_vw  | salesid    | integer                     | 1       |
| public      | sales_vw  | listid     | integer                     | 2       |
| public      | sales_vw  | sellerid   | integer                     | 3       |
| public      | sales_vw  | buyerid    | integer                     | 4       |
| public      | sales_vw  | eventid    | integer                     | 5       |
| public      | sales_vw  | dateid     | smallint                    | 6       |
| public      | sales_vw  | qtysold    | smallint                    | 7       |
| public      | sales_vw  | pricepaid  | numeric(8,2)                | 8       |
| public      | sales_vw  | commission | numeric(8,2)                | 9       |
| public      | sales_vw  | saletime   | timestamp without time zone | 10      |

## PG\_GET\_GRANTEE\_BY\_IAM\_ROLE

Returns all users and groups granted a specified IAM role.

## Syntax

```
pg_get_grantee_by_iam_role('iam_role_arn')
```

## Arguments

*iam\_role\_arn*

The IAM role for which to return the users and groups that have been granted this role.

## Return type

VARCHAR

## Usage notes

The PG\_GET\_GRANTEE\_BY\_IAM\_ROLE function returns one row for each user or group. Each row contains the grantee name, grantee type, and granted privilege. The possible values for the grantee type are p for public, u for user, and g for group.

You must be a superuser to use this function.

## Example

The following example indicates that the IAM role Redshift-S3-Write is granted to group1 and reg\_user1. Users in group\_1 can specify the role only for COPY operations, and user reg\_user1 can specify the role only to perform UNLOAD operations.

```
select pg_get_grantee_by_iam_role('arn:aws:iam::123456789012:role/Redshift-S3-Write');
```

```
pg_get_grantee_by_iam_role

(group_1,g,COPY)
(reg_user1,u,UNLOAD)
```

The following example of the PG\_GET\_GRANTEE\_BY\_IAM\_ROLE function formats the result as a table.

```
select grantee, grantee_type, cmd_type FROM
pg_get_grantee_by_iam_role('arn:aws:iam::123456789012:role/Redshift-S3-Write')
res_grantee(grantee text, grantee_type text, cmd_type text) ORDER BY 1,2,3;
```

| grantee   | grantee_type | cmd_type |
|-----------|--------------|----------|
| group_1   | g            | COPY     |
| reg_user1 | u            | UNLOAD   |

## PG\_GET\_IAM\_ROLE\_BY\_USER

Returns all IAM roles and command privileges granted to a user.

## Syntax

```
pg_get_iam_role_by_user('name')
```

## Arguments

*name*

The name of the user for which to return IAM roles.

## Return type

VARCHAR

## Usage notes

The PG\_GET\_IAM\_ROLE\_BY\_USER function returns one row for each set of roles and command privileges. The row contains a comma-separated list with the user name, IAM role, and command.

A value of default in the result indicates that the user can specify any available role to perform the displayed command.

You must be a superuser to use this function.

## Example

The following example indicates that user reg\_user1 can specify any available IAM role to perform COPY operations. The user can also specify the Redshift-S3-Write role for UNLOAD operations.

```
select pg_get_iam_role_by_user('reg_user1');
```

```
pg_get_iam_role_by_user

(reg_user1,default,COPY)
(reg_user1,arn:aws:iam::123456789012:role/Redshift-S3-Write,COPY|UNLOAD)
```

The following example of the PG\_GET\_IAM\_ROLE\_BY\_USER function formats the result as a table.

```
select username, iam_role, cmd FROM pg_get_iam_role_by_user('reg_user1')
res_iam_role(username text, iam_role text, cmd text);
```

| username  | iam_role                                        | cmd  |
|-----------|-------------------------------------------------|------|
| reg_user1 | default                                         | None |
| reg_user1 | arn:aws:iam::123456789012:role/Redshift-S3-Read | COPY |

## PG\_GET\_LATE\_BINDING\_VIEW\_COLS

Returns the column metadata for all late-binding views in the database. For more information, see [Late-binding views \(p. 853\)](#)

### Syntax

```
pg_get_late_binding_view_cols()
```

### Return type

VARCHAR

## Usage notes

The PG\_GET\_LATE\_BINDING\_VIEW\_COLS function returns one row for each column in late-binding views. The row contains a comma-separated list with the schema name, relation name, column name, data type, and column number.

## Example

The following example returns the column metadata for all late-binding views.

```
select pg_get_late_binding_view_cols();

pg_get_late_binding_view_cols

(public,myevent,eventname,"character varying(200)",1)
(public,sales_lbv,salesid,integer,1)
(public,sales_lbv,listid,integer,2)
(public,sales_lbv,sellerid,integer,3)
(public,sales_lbv,buyerid,integer,4)
(public,sales_lbv,eventid,integer,5)
(public,sales_lbv,dateid,smallint,6)
(public,sales_lbv,qtysold,smallint,7)
(public,sales_lbv,pricepaid,"numeric(8,2)",8)
(public,sales_lbv,commission,"numeric(8,2)",9)
(public,sales_lbv,saletime,"timestamp without time zone",10)
(public,event_lbv,eventid,integer,1)
(public,event_lbv,venueid,smallint,2)
(public,event_lbv,catid,smallint,3)
(public,event_lbv,dateid,smallint,4)
(public,event_lbv,eventname,"character varying(200)",5)
(public,event_lbv,starttime,"timestamp without time zone",6)
```

The following example returns the column metadata for all late-binding views in table format.

| view_schema | view_name | col_name   | col_type                      | col_num |
|-------------|-----------|------------|-------------------------------|---------|
| public      | sales_lbv | salesid    | integer                       | 1       |
| public      | sales_lbv | listid     | integer                       | 2       |
| public      | sales_lbv | sellerid   | integer                       | 3       |
| public      | sales_lbv | buyerid    | integer                       | 4       |
| public      | sales_lbv | eventid    | integer                       | 5       |
| public      | sales_lbv | dateid     | smallint                      | 6       |
| public      | sales_lbv | qtysold    | smallint                      | 7       |
| public      | sales_lbv | pricepaid  | numeric(8,2)                  | 8       |
| public      | sales_lbv | commission | numeric(8,2)                  | 9       |
| public      | sales_lbv | saletime   | timestamp without time zone   | 10      |
| public      | event_lbv | eventid    | integer                       | 1       |
| public      | event_lbv | venueid    | smallint                      | 2       |
| public      | event_lbv | catid      | smallint                      | 3       |
| public      | event_lbv | dateid     | smallint                      | 4       |
| public      | event_lbv | eventname  | "character varying(200)"      | 5       |
| public      | event_lbv | starttime  | "timestamp without time zone" | 6       |

## PG\_LAST\_COPY\_COUNT

Returns the number of rows that were loaded by the last COPY command run in the current session. PG\_LAST\_COPY\_COUNT is updated with the last COPY ID, which is the query ID of the last COPY that began the load process, even if the load failed. The query ID and COPY ID are updated when the COPY command begins the load process.

If the COPY fails because of a syntax error or because of insufficient privileges, the COPY ID is not updated and PG\_LAST\_COPY\_COUNT returns the count for the previous COPY. If no COPY commands were run in the current session, or if the last COPY failed during loading, PG\_LAST\_COPY\_COUNT returns 0. For more information, see [PG\\_LAST\\_COPY\\_ID \(p. 1400\)](#).

### Syntax

```
pg_last_copy_count()
```

## Return type

Returns BIGINT.

## Example

The following query returns the number of rows loaded by the latest COPY command in the current session.

```
select pg_last_copy_count();
pg_last_copy_count

192497
(1 row)
```

## PG\_LAST\_COPY\_ID

Returns the query ID of the most recently completed COPY command in the current session. If no COPY commands have been run in the current session, PG\_LAST\_COPY\_ID returns -1.

The value for PG\_LAST\_COPY\_ID is updated when the COPY command begins the load process. If the COPY fails because of invalid load data, the COPY ID is updated, so you can use PG\_LAST\_COPY\_ID when you query STL\_LOAD\_ERRORS table. If the COPY transaction is rolled back, the COPY ID is not updated.

The COPY ID is not updated if the COPY command fails because of an error that occurs before the load process begins, such as a syntax error, access error, invalid credentials, or insufficient privileges. The COPY ID is not updated if the COPY fails during the analyze compression step, which begins after a successful connection, but before the data load. COPY performs compression analysis when the COMPUPDATE parameter is set to ON or when the target table is empty and all the table columns either have RAW encoding or no encoding.

## Syntax

```
pg_last_copy_id()
```

## Return type

Returns an integer.

## Example

The following query returns the query ID of the latest COPY command in the current session.

```
select pg_last_copy_id();
pg_last_copy_id

5437
(1 row)
```

The following query joins STL\_LOAD\_ERRORS to STL\_LOADERROR\_DETAIL to view the details errors that occurred during the most recent load in the current session:

```
select d.query, substring(d.filename,14,20),
d.line_number as line,
substring(d.value,1,16) as value,
```

```

substring(le.err_reason,1,48) as err_reason
from stl_loaderror_detail d, stl_load_errors le
where d.query = le.query
and d.query = pg_last_copy_id();

query | substring | line | value | err_reason
-----+-----+-----+-----+
558| allusers_pipe.txt | 251 | 251 | String contains invalid or
 | | | unsupported UTF8 code
558| allusers_pipe.txt | 251 | ZRU29FGR | String contains invalid or
 | | | unsupported UTF8 code
558| allusers_pipe.txt | 251 | Kaitlin | String contains invalid or
 | | | unsupported UTF8 code
558| allusers_pipe.txt | 251 | Walter | String contains invalid or
 | | | unsupported UTF8 code

```

## PG\_LAST\_UNLOAD\_ID

Returns the query ID of the most recently completed UNLOAD command in the current session. If no UNLOAD commands have been run in the current session, PG\_LAST\_UNLOAD\_ID returns -1.

The value for PG\_LAST\_UNLOAD\_ID is updated when the UNLOAD command begins the load process. If the UNLOAD fails because of invalid load data, the UNLOAD ID is updated, so you can use the UNLOAD ID for further investigation. If the UNLOAD transaction is rolled back, the UNLOAD ID is not updated.

The UNLOAD ID is not updated if the UNLOAD command fails because of an error that occurs before the load process begins, such as a syntax error, access error, invalid credentials, or insufficient privileges.

### Syntax

```
PG_LAST_UNLOAD_ID()
```

### Return type

Returns an integer.

### Example

The following query returns the query ID of the latest UNLOAD command in the current session.

```

select PG_LAST_UNLOAD_ID();

PG_LAST_UNLOAD_ID

 5437
(1 row)

```

## PG\_LAST\_QUERY\_ID

Returns the query ID of the most recently completed query in the current session. If no queries have been run in the current session, PG\_LAST\_QUERY\_ID returns -1. PG\_LAST\_QUERY\_ID does not return the query ID for queries that run exclusively on the leader node. For more information, see [Leader node-only functions \(p. 1020\)](#).

### Syntax

```
pg_last_query_id()
```

## Return type

Returns an integer.

## Example

The following query returns the ID of the latest query completed in the current session.

```
select pg_last_query_id();
```

Results are the following.

```
pg_last_query_id

5437
(1 row)
```

The following query returns the query ID and text of the most recently completed query in the current session.

```
select query, trim(querytxt) as sqlquery
from stl_query
where query = pg_last_query_id();
```

Results are the following.

```
query | sqlquery
-----+
5437 | select name, loadtime from stl_file_scan where loadtime > 1000000;
(1 rows)
```

## PG\_LAST\_UNLOAD\_COUNT

Returns the number of rows that were unloaded by the last UNLOAD command completed in the current session. PG\_LAST\_UNLOAD\_COUNT is updated with the query ID of the last UNLOAD, even if the operation failed. The query ID is updated when the UNLOAD is completed. If the UNLOAD fails because of a syntax error or because of insufficient privileges, PG\_LAST\_UNLOAD\_COUNT returns the count for the previous UNLOAD. If no UNLOAD commands were completed in the current session, or if the last UNLOAD failed during the unload operation, PG\_LAST\_UNLOAD\_COUNT returns 0.

## Syntax

```
pg_last_unload_count()
```

## Return type

Returns BIGINT.

## Example

The following query returns the number of rows unloaded by the latest UNLOAD command in the current session.

```
select pg_last_unload_count();
```

```
pg_last_unload_count

192497
(1 row)
```

## SLICE\_NUM Function

Returns an integer corresponding to the slice number in the cluster where the data for a row is located. SLICE\_NUM takes no parameters.

### Syntax

```
SLICE_NUM()
```

### Return type

The SLICE\_NUM function returns an integer.

### Examples

The following example shows which slices contain data for the first ten EVENT rows in the EVENTS table:

```
select distinct eventid, slice_num() from event order by eventid limit 10;

eventid | slice_num
-----+-----
 1 | 1
 2 | 2
 3 | 3
 4 | 0
 5 | 1
 6 | 2
 7 | 3
 8 | 0
 9 | 1
10 | 2
(10 rows)
```

The following example returns a code (10000) to show that a query without a FROM statement runs on the leader node:

```
select slice_num();
slice_num

10000
(1 row)
```

## USER

Synonym for CURRENT\_USER. See [CURRENT\\_USER \(p. 1389\)](#).

## ROLE\_IS\_MEMBER\_OF

Returns true if the role is a member of another role. Superusers can check the membership of all roles. Regular users who have the ACCESS SYSTEM TABLE permission can check all users' membership. Otherwise, regular users can only check roles to which they have access. Amazon Redshift errors out if the provided roles don't exist or the current user doesn't have access to the role.

## Syntax

```
role_is_member_of(role_name, granted_role_name)
```

## Arguments

*role\_name*

The name of the role.

*granted\_role\_name*

The name of the granted role.

## Return type

Returns a BOOLEAN.

## Example

The following query confirms that the role isn't a member of role1 nor role2.

```
SELECT role_is_member_of('role1', 'role2');

role_is_member_of

False
```

## USER\_IS\_MEMBER\_OF

Returns true if the user is a member of a role or group. Superusers can check the membership of all users. Regular users who are members of the sys:secadmin or sys:superuser role can check all users' membership. Otherwise, regular users can only check themselves. Amazon Redshift sends an error if the provided identities don't exist or the current user doesn't have access to the role.

## Syntax

```
user_is_member_of(user_name, role_name | group_name)
```

## Arguments

*user\_name*

The name of the user.

*role\_name*

The name of the role.

*group\_name*

The name of the group.

## Return type

Returns a BOOLEAN.

## Example

The following query confirms that the user isn't a member of role1.

```
SELECT user_is_member_of('reguser', 'role1');

user_is_member_of

 False
```

## VERSION

The VERSION function returns details about the currently installed release, with specific Amazon Redshift version information at the end.

**Note**

This is a leader-node function. This function returns an error if it references a user-created table, an STL or STV system table, or an SVV or SVL system view.

### Syntax

```
VERSION()
```

### Return type

Returns a CHAR or VARCHAR string.

### Examples

The following example shows the cluster version information of the current cluster:

```
select version();
```

```
version
```

```
PostgreSQL 8.0.2 on i686-pc-linux-gnu, compiled by GCC gcc (GCC) 3.4.2 20041017 (Red Hat 3.4.2-6.fc3), Redshift 1.0.12103
```

Where 1.0.12103 is the cluster version number.

**Note**

To force your cluster to update to the latest cluster version, adjust your [maintenance window](#).

## Reserved words

The following is a list of Amazon Redshift reserved words. You can use the reserved words with delimited identifiers (double quotation marks).

**Note**

While START and CONNECT are not reserved words, use delimited identifiers or AS if you're using START and CONNECT as table aliases in your query to avoid failure at runtime.

For more information, see [Names and identifiers \(p. 571\)](#).

```
AES128
```

```
AES256
ALL
ALLOWOVERWRITE
ANALYSE
ANALYZE
AND
ANY
ARRAY
AS
ASC
AUTHORIZATION
AZ64
BACKUP
BETWEEN
BINARY
BLANKSASNULL
BOTH
BYTEDICT
BZIP2
CASE
CAST
CHECK
COLLATE
COLUMN
CONNECT
CONSTRAINT
CREATE
CREDENTIALS
CROSS
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURRENT_USER_ID
DEFAULT
DEFERRABLE
DEFLATE
DEFrag
DELTA
DELTA32K
DESC
DISABLE
DISTINCT
DO
ELSE
EMPTYASNULL
ENABLE
ENCODE
ENCRYPT
ENCRYPTION
END
EXCEPT
EXPLICIT
FALSE
FOR
FOREIGN
FREEZE
FROM
FULL
GLOBALDICT256
GLOBALDICT64K
GRANT
GROUP
GZIP
HAVING
IDENTITY
```

|                |
|----------------|
| IGNORE         |
| ILIKE          |
| IN             |
| INITIALLY      |
| INNER          |
| INTERSECT      |
| INTERVAL       |
| INTO           |
| IS             |
| ISNULL         |
| JOIN           |
| LANGUAGE       |
| LEADING        |
| LEFT           |
| LIKE           |
| LIMIT          |
| LOCALTIME      |
| LOCALTIMESTAMP |
| LUN            |
| LUNS           |
| LZO            |
| LZOP           |
| MINUS          |
| MOSTLY16       |
| MOSTLY32       |
| MOSTLY8        |
| NATURAL        |
| NEW            |
| NOT            |
| NOTNULL        |
| NULL           |
| NULLS          |
| OFF            |
| OFFLINE        |
| OFFSET         |
| OID            |
| OLD            |
| ON             |
| ONLY           |
| OPEN           |
| OR             |
| ORDER          |
| OUTER          |
| OVERLAPS       |
| PARALLEL       |
| PARTITION      |
| PERCENT        |
| PERMISSIONS    |
| PIVOT          |
| PLACING        |
| PRIMARY        |
| RAW            |
| READRATIO      |
| RECOVER        |
| REFERENCES     |
| REJECTLOG      |
| RESORT         |
| RESPECT        |
| RESTORE        |
| RIGHT          |
| SELECT         |
| SESSION_USER   |
| SIMILAR        |
| SNAPSHOT       |
| SOME           |
| SYSDATE        |

|                 |
|-----------------|
| SYSTEM          |
| START           |
| TABLE           |
| TAG             |
| TDES            |
| TEXT255         |
| TEXT32K         |
| THEN            |
| TIMESTAMP       |
| TO              |
| TOP             |
| TRAILING        |
| TRUE            |
| TRUNCATECOLUMNS |
| UNION           |
| UNIQUE          |
| UNNEST          |
| UNPIVOT         |
| USER            |
| USING           |
| VERBOSE         |
| WALLET          |
| WHEN            |
| WHERE           |
| WITH            |
| WITHOUT         |

# System tables and views reference

## Topics

- [System tables and views \(p. 1409\)](#)
- [Types of system tables and views \(p. 1409\)](#)
- [Visibility of data in system tables and views \(p. 1410\)](#)
- [SVV metadata views \(p. 1411\)](#)
- [SYS monitoring views \(p. 1465\)](#)
- [System monitoring \(provisioned only\) \(p. 1494\)](#)
- [System catalog tables \(p. 1670\)](#)

## System tables and views

Amazon Redshift has many system tables and views that contain information about how the system is functioning. You can query these system tables and views the same way that you would query any other database tables. This section shows some sample system table queries and explains:

- How different types of system tables and views are generated
- What types of information you can obtain from these tables
- How to join Amazon Redshift system tables to catalog tables
- How to manage the growth of system table log files

Some system tables can only be used by AWS staff for diagnostic purposes. The following sections discuss the system tables that can be queried for useful information by system administrators or other database users.

### Note

System tables are not included in automated or manual cluster backups (snapshots). STL system views retain seven days of log history. Retaining logs doesn't require any customer action, but if you want to store log data for more than 7 days, you have to periodically copy it to other tables or unload it to Amazon S3.

## Types of system tables and views

There are several types of system tables and views:

- SVV views contain information about database objects with references to transient STV tables.
- SYS views are used to monitor query and workload usage for provisioned clusters and serverless workgroups.
- STL views are generated from logs that have been persisted to disk to provide a history of the system.
- STV tables are virtual system tables that contain snapshots of the current system data. They are based on transient in-memory data and are not persisted to disk-based logs or regular tables.
- SVCS views provide details about queries on both the main and concurrency scaling clusters.
- SVL views provide details about queries on main clusters.

System tables and views do not use the same consistency model as regular tables. It is important to be aware of this issue when querying them, especially for STV tables and SVV views. For example, given a regular table t1 with a column c1, you would expect that the following query to return no rows:

```
select * from t1
where c1 > (select max(c1) from t1)
```

However, the following query against a system table might well return rows:

```
select * from stv_exec_state
where currenttime > (select max(currenttime) from stv_exec_state)
```

The reason this query might return rows is that currenttime is transient and the two references in the query might not return the same value when evaluated.

On the other hand, the following query might well return no rows:

```
select * from stv_exec_state
where currenttime = (select max(currenttime) from stv_exec_state)
```

## Visibility of data in system tables and views

There are two classes of visibility for data in system tables and views: visible to users and visible to superusers.

Only users with superuser privileges can see the data in those tables that are in the superuser-visible category. Regular users can see data in the user-visible tables. To give a regular user access to superuser-visible tables, [GRANT \(p. 887\)](#) SELECT privilege on that table to the regular user.

By default, in most user-visible tables, rows generated by another user are invisible to a regular user. If a regular user is given unrestricted [SYSLOG ACCESS \(p. 660\)](#), that user can see all rows in user-visible tables, including rows generated by another user. For more information, see [ALTER USER \(p. 659\)](#) or [CREATE USER \(p. 846\)](#). All rows in SVV\_TRANSACTIONS are visible to all users.

For metadata views, Amazon Redshift doesn't allow visibility to users that are granted unrestricted SYSLOG ACCESS.

### Note

Giving a user unrestricted access to system tables gives the user visibility to data generated by other users. For example, STL\_QUERY and STL\_QUERY\_TEXT contain the full text of INSERT, UPDATE, and DELETE statements, which might contain sensitive user-generated data.

A superuser can see all rows in all tables. To give a regular user access to superuser-visible tables, [GRANT \(p. 887\)](#) SELECT privilege on that table to the regular user.

## Filtering system-generated queries

The query-related system tables and views, such as SVL\_QUERY\_SUMMARY, SVL\_QLOG, and others, usually contain a large number of automatically generated statements that Amazon Redshift uses to monitor the status of the database. These system-generated queries are visible to a superuser, but are seldom useful. To filter them out when selecting from a system table or system view that uses the userid column, add the condition userid > 1 to the WHERE clause. For example:

```
select * from svl_query_summary where userid > 1
```

# SVV metadata views

SVV views are system views in Amazon Redshift that contain information about database objects.

## Note

Amazon Redshift reports a WARNING, not an ERROR, if a database response fails for any reason. Amazon Redshift doesn't send ERROR messages when you're querying objects in a datashare.

## Topics

- [SVV\\_ALL\\_COLUMNS \(p. 1412\)](#)
- [SVV\\_ALL\\_SCHEMAS \(p. 1413\)](#)
- [SVV\\_ALL\\_TABLES \(p. 1414\)](#)
- [SVV\\_ALTER\\_TABLE\\_RECOMMENDATIONS \(p. 1415\)](#)
- [SVV\\_ATTACHED\\_MASKING\\_POLICY \(p. 1416\)](#)
- [SVV\\_COLUMNS \(p. 1418\)](#)
- [SVV\\_COLUMN\\_PRIVILEGES \(p. 1419\)](#)
- [SVV\\_DATABASE\\_PRIVILEGES \(p. 1420\)](#)
- [SVV\\_DATASHARE\\_PRIVILEGES \(p. 1421\)](#)
- [SVV\\_DATASHARES \(p. 1422\)](#)
- [SVV\\_DATASHARE\\_CONSUMERS \(p. 1423\)](#)
- [SVV\\_DATASHARE\\_OBJECTS \(p. 1424\)](#)
- [SVV\\_DEFAULT\\_PRIVILEGES \(p. 1425\)](#)
- [SVV\\_DISKUSAGE \(p. 1426\)](#)
- [SVV\\_EXTERNAL\\_COLUMNS \(p. 1428\)](#)
- [SVV\\_EXTERNAL\\_DATABASES \(p. 1429\)](#)
- [SVV\\_EXTERNAL\\_PARTITIONS \(p. 1429\)](#)
- [SVV\\_EXTERNAL\\_SCHEMAS \(p. 1430\)](#)
- [SVV\\_EXTERNAL\\_TABLES \(p. 1431\)](#)
- [SVV\\_FUNCTION\\_PRIVILEGES \(p. 1432\)](#)
- [SVV\\_IAM\\_PRIVILEGES \(p. 1433\)](#)
- [SVV\\_INTERLEAVED\\_COLUMNS \(p. 1434\)](#)
- [SVV\\_IDENTITY\\_PROVIDERS \(p. 1434\)](#)
- [SVV\\_LANGUAGE\\_PRIVILEGES \(p. 1435\)](#)
- [SVV\\_MASKING\\_POLICY \(p. 1436\)](#)
- [SVV\\_ML\\_MODEL\\_INFO \(p. 1437\)](#)
- [SVV\\_ML\\_MODEL\\_PRIVILEGES \(p. 1438\)](#)
- [SVV\\_MV\\_DEPENDENCY \(p. 1439\)](#)
- [SVV\\_MV\\_INFO \(p. 1439\)](#)
- [SVV\\_QUERY\\_INFLIGHT \(p. 1441\)](#)
- [SVV\\_QUERY\\_STATE \(p. 1442\)](#)
- [SVV\\_REDSHIFT\\_COLUMNS \(p. 1444\)](#)
- [SVV\\_REDSHIFT\\_DATABASES \(p. 1445\)](#)
- [SVV\\_REDSHIFT\\_FUNCTIONS \(p. 1446\)](#)
- [SVV\\_REDSHIFT\\_SCHEMAS \(p. 1447\)](#)
- [SVV\\_REDSHIFT\\_TABLES \(p. 1448\)](#)
- [SVV\\_RELATION\\_PRIVILEGES \(p. 1449\)](#)

- [SVV\\_RLS\\_APPLIED\\_POLICY \(p. 1449\)](#)
- [SVV\\_RLS\\_ATTACHED\\_POLICY \(p. 1450\)](#)
- [SVV\\_RLS\\_POLICY \(p. 1451\)](#)
- [SVV\\_RLS\\_RELATION \(p. 1452\)](#)
- [SVV\\_ROLES \(p. 1453\)](#)
- [SVV\\_ROLE\\_GRANTS \(p. 1454\)](#)
- [SVV\\_SCHEMA\\_QUOTA\\_STATE \(p. 1454\)](#)
- [SVV\\_SCHEMA\\_PRIVILEGES \(p. 1455\)](#)
- [SVV\\_SYSTEM\\_PRIVILEGES \(p. 1456\)](#)
- [SVV\\_TABLES \(p. 1456\)](#)
- [SVV\\_TABLE\\_INFO \(p. 1457\)](#)
- [SVV\\_TRANSACTIONS \(p. 1460\)](#)
- [SVV\\_USER\\_INFO \(p. 1461\)](#)
- [SVV\\_USER\\_GRANTS \(p. 1462\)](#)
- [SVV\\_VACUUM\\_PROGRESS \(p. 1463\)](#)
- [SVV\\_VACUUM\\_SUMMARY \(p. 1464\)](#)

## SVV\_ALL\_COLUMNS

Use SVV\_ALL\_COLUMNS to view a union of columns from Amazon Redshift tables as shown in SVV\_REDSHIFT\_COLUMNS and the consolidated list of all external columns from all external tables. For information about Amazon Redshift columns, see [SVV\\_REDSHIFT\\_COLUMNS \(p. 1444\)](#).

SVV\_ALL\_COLUMNS is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

### Table columns

| Column name              | Data type     | Description                                                                            |
|--------------------------|---------------|----------------------------------------------------------------------------------------|
| database_name            | varchar(128)  | The name of the database.                                                              |
| schema_name              | varchar(128)  | The name of the schema.                                                                |
| table_name               | varchar(128)  | The name of the table.                                                                 |
| column_name              | varchar(128)  | The name of the column.                                                                |
| ordinal_position         | integer       | The position of the column in the table.                                               |
| column_default           | varchar(4000) | The default value of the column.                                                       |
| is_nullable              | varchar(3)    | A value that indicates whether the column is nullable. Possible values are yes and no. |
| data_type                | varchar(128)  | The data type of the column.                                                           |
| character_maximum_length | integer       | The maximum number of characters in the column.                                        |
| numeric_precision        | integer       | The numeric precision.                                                                 |

| Column name   | Data type    | Description        |
|---------------|--------------|--------------------|
| numeric_scale | integer      | The numeric scale. |
| remarks       | varchar(256) | Remarks.           |

## Sample queries

The following example returns the output of SVV\_ALL\_COLUMNS.

```
SELECT *
FROM svv_all_columns
WHERE database_name = 'tickit_db'
 AND TABLE_NAME = 'ticket_sales_redshift'
ORDER BY COLUMN_NAME,
 SCHEMA_NAME
LIMIT 5;

database_name | schema_name | table_name | column_name | ordinal_position |
column_default | is_nullable | data_type | character_maximum_length | numeric_precision |
numeric_scale | remarks
-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+
-----+-----+
 tickit_db | public | ticket_sales_redshift | buyerid | 4 | |
 0 | NO | integer | | 32 | |
 tickit_db | public | ticket_sales_redshift | commission | 9 | |
 | | YES | numeric | | 8 | |
 | | | | | 2 | |
 tickit_db | public | ticket_sales_redshift | dateid | 7 | |
 | | NO | smallint | | 16 | |
 0 | | | | 16 | |
 tickit_db | public | ticket_sales_redshift | eventid | 5 | |
 | | NO | integer | | 32 | |
 0 | | | | 32 | |
 tickit_db | public | ticket_sales_redshift | listid | 2 | |
 | | NO | integer | | 32 | |
 0 | | | | 32 | |
```

## SVV\_ALL\_SCHEMAS

Use SVV\_ALL\_SCHEMAS to view a union of Amazon Redshift schemas as shown in SVV\_REDSHIFT\_SCHEMAS and the consolidated list of all external schemas from all databases. For more information about Amazon Redshift schemas, see [SVV\\_REDSHIFT\\_SCHEMAS \(p. 1447\)](#).

SVV\_ALL\_SCHEMAS is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

### Table columns

| Column name   | Data type    | Description                                       |
|---------------|--------------|---------------------------------------------------|
| database_name | varchar(128) | The name of the database where the schema exists. |
| schema_name   | varchar(128) | The name of the schema.                           |

| Column name     | Data type    | Description                                                                                  |
|-----------------|--------------|----------------------------------------------------------------------------------------------|
| schema_owner    | integer      | The user ID of the schema owner.                                                             |
| schema_type     | varchar(128) | The type of the schema. Possible values are external, local, and shared schemas.             |
| schema_acl      | varchar(128) | The string that defines the permissions for the specified user or user group for the schema. |
| source_database | varchar(128) | The name of the source database for external schema.                                         |
| schema_option   | varchar(256) | The options of the schema. This is an external schema attribute.                             |

## Sample query

The following example returns the output of SVV\_ALL\_SCHEMAS.

```
SELECT *
FROM svv_all_schemas
WHERE database_name = 'tickit_db'
ORDER BY database_name,
SCHEMA_NAME;

database_name | schema_name | schema_owner | schema_type | schema_acl |
source_database | schema_option
-----+-----+-----+-----+-----+
+-----+-----+
| tickit_db | public | 1 | shared | |
```

## SVV\_ALL\_TABLES

Use SVV\_ALL\_TABLES to view a union of Amazon Redshift tables as shown in SVV\_REDSHIFT\_TABLES and the consolidated list of all external tables from all external schemas. For information about Amazon Redshift tables, see [SVV\\_REDSHIFT\\_TABLES \(p. 1448\)](#).

SVV\_ALL\_TABLES is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

### Table columns

| Column name   | Data type    | Description                                      |
|---------------|--------------|--------------------------------------------------|
| database_name | varchar(128) | The name of the database where the table exists. |
| schema_name   | varchar(128) | The schema name for the table.                   |
| table_name    | varchar(128) | The name of the table.                           |

| Column name | Data type    | Description                                                                                        |
|-------------|--------------|----------------------------------------------------------------------------------------------------|
| table_acl   | varchar(128) | The string that defines the permission for the specified user or user group for the table.         |
| table_type  | varchar(128) | The type of the table. Possible values are views, base tables, external tables, and shared tables. |
| remarks     | varchar(256) | Remarks.                                                                                           |

## Sample queries

The following example returns the output of SVV\_ALL\_TABLES.

```
SELECT *
FROM svv_all_tables
WHERE database_name = 'tickit_db'
ORDER BY TABLE_NAME,
 SCHEMA_NAME
LIMIT 5;
```

| database_name | schema_name | table_name               | table_type | table_acl | remarks |
|---------------|-------------|--------------------------|------------|-----------|---------|
| tickit_db     | public      | tickit_category_redshift | TABLE      |           |         |
| tickit_db     | public      | tickit_date_redshift     | TABLE      |           |         |
| tickit_db     | public      | tickit_event_redshift    | TABLE      |           |         |
| tickit_db     | public      | tickit_listing_redshift  | TABLE      |           |         |
| tickit_db     | public      | tickit_sales_redshift    | TABLE      |           |         |

## SVV\_ALTER\_TABLE\_RECOMMENDATIONS

Records the current Amazon Redshift Advisor recommendations for tables. This view shows recommendations for all tables, whether they are defined for automatic optimization or not. To view if a table is defined for automatic optimization, see [SVV\\_TABLE\\_INFO \(p. 1457\)](#). Entries appear only for tables visible in the current session's database. After a recommendation has been applied (either by Amazon Redshift or by you), it no longer appears in the view.

SVV\_ALTER\_TABLE\_RECOMMENDATIONS is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type      | Description                                                                                                                                                                                                                                           |
|-------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| type        | character(30)  | The type of recommendation. Possible values are distkey and sortkey.                                                                                                                                                                                  |
| database    | character(128) | The database name.                                                                                                                                                                                                                                    |
| table_id    | integer        | The table identifier.                                                                                                                                                                                                                                 |
| group_id    | integer        | The group number of a set of recommendations. All recommendations in a group should be applied to see the maximum benefit. Possible values are -1 for a sort key recommendation, and a number larger than zero for a distribution key recommendation. |

| Column name   | Data type       | Description                                                                                                                                                                     |
|---------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ddl           | character(1024) | SQL statement that must run to apply the recommendation.                                                                                                                        |
| auto_eligible | character(1)    | The value indicates if the recommendation is eligible for Amazon Redshift to automatically run automatically. If this value is t, then the indication is true, if f then false. |

## Sample queries

In the following example, the rows in the result show recommendations for distribution key and sort key. The rows also show whether the recommendations are eligible for Amazon Redshift to automatically apply them.

```
select type, database, table_id, group_id, ddl, auto_eligible
from svv_alter_table_recommendations;
```

|           |          |          |          |                                                                       |
|-----------|----------|----------|----------|-----------------------------------------------------------------------|
| type      | database | table_id | group_id | ddl                                                                   |
|           |          |          |          | auto_eligible                                                         |
| diststyle | db0      | 117884   | 2        | ALTER TABLE "sch"."dp21235_tbl_1" ALTER                               |
|           |          |          |          | DISTSTYLE KEY DISTKEY "c0"                                            |
|           |          |          |          | f                                                                     |
| diststyle | db0      | 117892   | 2        | ALTER TABLE "sch"."dp21235_tbl_1" ALTER                               |
|           |          |          |          | DISTSTYLE KEY DISTKEY "c0"                                            |
|           |          |          |          | f                                                                     |
| diststyle | db0      | 117885   | 1        | ALTER TABLE "sch"."catalog_returns"                                   |
|           |          |          |          | ALTER DISTSTYLE KEY DISTKEY "cr_sold_date_sk", ALTER COMPOUND SORTKEY |
|           |          |          |          | ("cr_sold_date_sk", "cr_returned_time_sk")   t                        |
| sortkey   | db0      | 117890   | -1       | ALTER TABLE "sch"."customer_addresses" ALTER                          |
|           |          |          |          | COMPOUND SORTKEY ("ca_address_sk")                                    |
|           |          |          |          | t                                                                     |

## SVV\_ATTACHED\_MASKING\_POLICY

Use SVV\_ATTACHED\_MASKING\_POLICY to view all the relations and roles/users with policies attached on the currently connected database.

Only users with the [sys:secadmin](#) role can view SVV\_ATTACHED\_MASKING\_POLICY. Regular users will see 0 rows.

## Table columns

| Column name | Data type | Description                                              |
|-------------|-----------|----------------------------------------------------------|
| policy_name | text      | The name of the masking policy attached to the table.    |
| schema_name | text      | The schema of the table to which the policy is attached. |
| table_name  | text      | The name of the table to which the policy is attached.   |

| Column name    | Data type | Description                                                                     |
|----------------|-----------|---------------------------------------------------------------------------------|
| table_type     | text      | The type of the table to which the policy is attached.                          |
| grantor        | text      | The name of the user that attached the policy.                                  |
| grantee        | text      | The name of the user/role to whom the policy is attached.                       |
| grantee_type   | text      | The type of grantee. This can be <i>role</i> , <i>user</i> , or <i>public</i> . |
| priority       | int       | The priority of the attached policy.                                            |
| input_columns  | text      | The input column attributes of the attached policy.                             |
| output_columns | text      | The output column attributes of the attached policy.                            |

## Internal functions

SVV\_ATTACHED\_MASKING\_POLICY supports the following internal functions:

### [mask\\_get\\_policy\\_for\\_role\\_on\\_column](#)

Get the highest priority policy that applies to a given column/role pair.

#### Syntax

```
mask_get_policy_for_role_on_column
 (relschema,
 relname,
 colname,
 rolename);
```

#### Parameters

*relschema*

The name of the schema the policy is in.

*relname*

The name of the table the policy is in.

*colname*

The name of the column the policy is attached to.

*rolename*

The name of the role the policy is attached to.

### [mask\\_get\\_policy\\_for\\_user\\_on\\_column](#)

Get the highest priority policy that applies to a given column/user pair.

## Syntax

```
mask_get_policy_for_user_on_column
 (relschema,
 relname,
 colname,
 username);
```

## Parameters

*relschema*

The name of the schema the policy is in.

*relname*

The name of the table the policy is in.

*colname*

The name of the column the policy is attached to.

*rolename*

The name of the user the policy is attached to.

## SVV\_COLUMNS

Use SVV\_COLUMNS to view catalog information about the columns of local and external tables and views, including [late-binding views \(p. 853\)](#).

SVV\_COLUMNS is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

The SVV\_COLUMNS view joins table metadata from the [System catalog tables \(p. 1670\)](#) (tables with a PG prefix) and the [SVV\\_EXTERNAL\\_COLUMNS \(p. 1428\)](#) system view. The system catalog tables describe Amazon Redshift database tables. SVV\_EXTERNAL\_COLUMNS describes external tables that are used with Amazon Redshift Spectrum.

All users can see all rows from the system catalog tables. Regular users can see column definitions from the SVV\_EXTERNAL\_COLUMNS view only for external tables to which they have been granted access. Although regular users can see table metadata in the system catalog tables, they can only select data from the user-defined tables if they own the table or have been granted access.

## Table columns

| Column name      | Data type | Description                                 |
|------------------|-----------|---------------------------------------------|
| table_catalog    | text      | The name of the catalog where the table is. |
| table_schema     | text      | The schema name for the table.              |
| table_name       | text      | The name of the table.                      |
| column_name      | text      | The name of the column.                     |
| ordinal_position | int       | The position of the column in the table.    |

| Column name              | Data type | Description                                            |
|--------------------------|-----------|--------------------------------------------------------|
| column_default           | text      | The default value of the column.                       |
| is_nullable              | text      | A value that indicates whether the column is nullable. |
| data_type                | text      | The data type of the column.                           |
| character_maximum_length | int       | The maximum number of characters in the column.        |
| numeric_precision        | int       | The numeric precision.                                 |
| numeric_precision_radix  | int       | The numeric precision radix.                           |
| numeric_scale            | int       | The numeric scale.                                     |
| datetime_precision       | int       | The datetime precision.                                |
| interval_type            | text      | The interval type.                                     |
| interval_precision       | text      | The interval precision.                                |
| character_set_catalog    | text      | The character set catalog.                             |
| character_set_schema     | text      | The character set schema.                              |
| character_set_name       | text      | The character set name.                                |
| collation_catalog        | text      | The collation catalog.                                 |
| collation_schema         | text      | The collation schema.                                  |
| collation_name           | text      | The collation name.                                    |
| domain_name              | text      | The domain name.                                       |
| remarks                  | text      | Remarks.                                               |

## SVV\_COLUMN\_PRIVILEGES

Use SVV\_COLUMN\_PRIVILEGES to view the column permissions that are explicitly granted to users, roles, and groups in the current database.

SVV\_COLUMN\_PRIVILEGES is visible to all users. Superusers can see all rows. Regular users who have the ACCESS SYSTEM TABLE permission can see all rows. Regular users can only see identities to which they have access or for which they are the identities owner.

### Table columns

| Column name    | Data type | Description                                                  |
|----------------|-----------|--------------------------------------------------------------|
| namespace_name | text      | The name of the namespace where a specified relation exists. |
| relation_name  | text      | The name of the relation.                                    |
| column_name    | text      | The name of the column.                                      |

| Column name    | Data type | Description                                                                |
|----------------|-----------|----------------------------------------------------------------------------|
| privilege_type | text      | The type of the permission. Possible values are SELECT or UPDATE.          |
| identity_id    | integer   | The ID of the identity. Possible values are user ID, role ID, or group ID. |
| identity_name  | text      | The name of the identity.                                                  |
| identity_type  | text      | The type of the identity. Possible values are user, role, group or public. |

## Sample query

The following example displays the result of the SVV\_COLUMN\_PRIVILEGES.

```
SELECT namespace_name,relation_name,COLUMN_NAME,privilege_type,identity_name,identity_type
FROM svv_column_privileges WHERE relation_name = 'lineitem';

namespace_name | relation_name | column_name | privilege_type | identity_name | identity_type
-----+-----+-----+-----+-----+-----+
public | lineitem | l_orderkey | SELECT | reguser | user
public | lineitem | l_orderkey | SELECT | role1 | role
public | lineitem | l_partkey | SELECT | reguser | user
public | lineitem | l_partkey | SELECT | role1 | role
```

## SVV\_DATABASE\_PRIVILEGES

Use SVV\_DATABASE\_PRIVILEGES to view the database permissions that are explicitly granted to users, roles, and groups in your Amazon Redshift cluster.

SVV\_DATABASE\_PRIVILEGES is visible to all users. Superusers can see all rows. Regular users who have the ACCESS SYSTEM TABLE permission can see all rows. Regular users can only see identities to which they have access or for which they are the identities owner.

## Table columns

| Column name    | Data type | Description                                                                                                                                         |
|----------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| database_name  | text      | The name of the database.                                                                                                                           |
| privilege_type | text      | The type of the permission. Possible values are USAGE, CREATE, or TEMP.                                                                             |
| identity_id    | integer   | The ID of the identity. Possible values are user ID, role ID, or group ID.                                                                          |
| identity_name  | text      | The name of the identity.                                                                                                                           |
| identity_type  | text      | The type of the identity. Possible values are user, role, group, or public.                                                                         |
| admin_option   | boolean   | A value that indicates whether the user can grant the permission to other users and roles. It is always false for the role and group identity type. |

## Sample query

The following example displays the result of the SVV\_DATABASE\_PRIVILEGES.

```
SELECT database_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_database_privileges
WHERE database_name = 'test_db';

database_name | privilege_type | identity_name | identity_type | admin_option
-----+-----+-----+-----+-----+
test_db | CREATE | reguser | user | False
test_db | CREATE | role1 | role | False
test_db | TEMP | public | public | False
test_db | TEMP | role1 | role | False
```

## SVV\_DATASHARE\_PRIVILEGES

Use SVV\_DATASHARE\_PRIVILEGES to view the datashare permissions that are explicitly granted to users, roles, and groups in your Amazon Redshift cluster.

SVV\_DATASHARE\_PRIVILEGES is visible to all users. Superusers can see all rows. Regular users who have the ACCESS SYSTEM TABLE permission can see all rows. Regular users can only see identities to which they have access or for which they are the identities owner.

## Table columns

| Column name    | Data type | Description                                                                                                                                         |
|----------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| datashare_name | text      | The name of the datashare.                                                                                                                          |
| privilege_type | text      | The type of the permission. Possible values are ALTER or SHARE.                                                                                     |
| identity_id    | integer   | The ID of the identity. Possible values are user ID, role ID, or group ID.                                                                          |
| identity_name  | text      | The name of the identity.                                                                                                                           |
| identity_type  | text      | The type of the identity. Possible values are user, role, group, or public.                                                                         |
| admin_option   | boolean   | A value that indicates whether the user can grant the permission to other users and roles. It is always false for the role and group identity type. |

## Sample query

The following example displays the result of the SVV\_DATASHARE\_PRIVILEGES.

```
SELECT datashare_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_datashare_privileges
WHERE datashare_name = 'demo_share';

datashare_name | privilege_type | identity_name | identity_type | admin_option
-----+-----+-----+-----+-----+
demo_share | ALTER | superuser | user | False
demo_share | ALTER | reguser | user | False
```

## SVV\_DATASHARES

Use SVV\_DATASHARES to view a list of datashares created on the cluster, and datashares shared with the cluster.

SVV\_DATASHARES is visible to superusers, owners of datashares, or users who have the ALTER or USAGE privilege on the datashare. Other users can't see any rows. For information on the ALTER or USAGE privilege, see [GRANT \(p. 887\)](#).

### Table columns

| Column name         | Data type                   | Description                                                                                     |
|---------------------|-----------------------------|-------------------------------------------------------------------------------------------------|
| share_name          | varchar(128)                | The name of a datashare.                                                                        |
| share_id            | integer                     | The ID of the datashare.                                                                        |
| share_owner         | integer                     | The owner of the datashare.                                                                     |
| source_database     | varchar(128)                | The source database for this datashare.                                                         |
| consumer_database   | varchar(128)                | The consumer database that is created from this datashare.                                      |
| share_type          | varchar(8)                  | The type of the datashare. Possible values are INBOUND and OUTBOUND.                            |
| createdate          | timestamp without time zone | The date when datashare was created.                                                            |
| is_publicaccessible | boolean                     | The property that specifies whether a datashare can be shared to a publicly accessible cluster. |
| share_acl           | varchar(256)                | The string that defines the permissions for the specified user or user group for the datashare. |
| producer_account    | varchar(16)                 | The ID for the datashare producer account.                                                      |
| producer_namespace  | varchar(64)                 | The unique cluster identifier for the datashare producer cluster.                               |
| managed_by          | varchar(64)                 | The property that specifies the AWS service that manages the datashare.                         |

### Sample query

The following example returns the output for SVV\_DATASHARES.

```
SELECT share_owner, source_database, share_type, is_publicaccessible
```

```
FROM svv_datashares
WHERE share_name LIKE 'tickit_datashare%'
AND source_database = 'dev';

share_owner | source_database | share_type | is_publicaccessible
-----+-----+-----+-----+
 100 | dev | OUTBOUND | True
(1 rows)
```

The following example returns the output for SVV\_DATASHARES for outbound datashares.

```
SELECT share_name, share_owner, btrim(source_database), btrim(consumer_database),
share_type, is_publicaccessible, share_acl, btrim(producer_account),
btrim(producer_namespace), btrim(managed_by) FROM svv_datashares WHERE share_type =
'OUTBOUND';

share_name | share_owner | source_database | consumer_database | share_type |
is_publicaccessible | share_acl | producer_account| producer_namespace |
managed_by
-----+-----+-----+-----+
-----+-----+-----+-----+
-----+-----+
salesshare | 1 | dev | | OUTBOUND |
True | | 123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
marketingshare | 1 | dev | | OUTBOUND |
True | | 123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
```

The following example returns the output for SVV\_DATASHARES for inbound datashares.

```
SELECT share_name, share_owner, btrim(source_database), btrim(consumer_database),
share_type, is_publicaccessible, share_acl, btrim(producer_account),
btrim(producer_namespace), btrim(managed_by) FROM svv_datashares WHERE share_type =
'INBOUND';

share_name | share_owner | source_database | consumer_database | share_type |
is_publicaccessible | share_acl | producer_account | producer_namespace |
managed_by
-----+-----+-----+-----+
-----+-----+-----+-----+
-----+-----+
salesshare | | 123456789012 | | INBOUND |
False | | 123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
marketingshare | | 123456789012 | | INBOUND |
False | | 123456789012 | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | ADX
```

## SVV\_DATASHARE\_CONSUMERS

Use SVV\_DATASHARE\_CONSUMERS to view a list of consumers for a datashare created on a cluster.

SVV\_DATASHARE\_CONSUMERS is visible to superusers, owners of datashares, or users who have the ALTER or USAGE permission on the datashare. Other users can't see any rows.

### Table columns

| Column name | Data type    | Description                |
|-------------|--------------|----------------------------|
| share_name  | varchar(128) | The name of the datashare. |

| Column name        | Data type                   | Description                                                      |
|--------------------|-----------------------------|------------------------------------------------------------------|
| consumer_account   | varchar(16)                 | The account ID for the datashare consumer.                       |
| consumer_namespace | varchar(64)                 | The unique cluster identifier of the datashare consumer cluster. |
| share_date         | timestamp without time zone | The date that the datashare was shared.                          |

## Sample query

The following example returns the output for SVV\_DATASHARE\_CONSUMERS.

```
SELECT count(*)
FROM svv_datashare_consumers
WHERE share_name LIKE 'tickit_datashare%';
1
```

## SVV\_DATASHARE\_OBJECTS

Use SVV\_DATASHARE\_OBJECTS to view a list of objects in all datashares created on the cluster or shared with the cluster.

SVV\_DATASHARE\_OBJECTS is visible only to superusers. Other users can't see any rows.

## Table columns

| Column name        | Data type    | Description                                                                                                                        |
|--------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------|
| share_type         | varchar(8)   | The type of the specified datashare. Possible values are OUTBOUND and INBOUND.                                                     |
| share_name         | varchar(128) | The name of the datashare.                                                                                                         |
| object_type        | varchar(64)  | The type of a specified object. Possible values are schemas, tables, views, late binding views, materialized views, and functions. |
| object_name        | varchar(512) | The name of the object. The object name extends to include the schema name, such as schema1.t1.                                    |
| producer_account   | varchar(16)  | The ID for the datashare producer account.                                                                                         |
| producer_namespace | varchar(64)  | The unique cluster identifier for the datashare producer cluster.                                                                  |

| Column name | Data type | Description                                                                                                                                                                                                                                                     |
|-------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| include_new | boolean   | The property that specifies whether to add any future tables, views, or SQL user-defined functions (UDFs) created in the specified schema to the datashare. This parameter is only relevant for OUTBOUND datashares and only for schema types in the datashare. |

## Sample query

The following examples return the output for SVV\_DATASHARE\_OBJECTS.

```
SELECT share_type,
 btrim(share_name)::varchar(16) AS share_name,
 object_type,
 object_name
 FROM svv_datashare_objects
 WHERE share_name LIKE 'tickit_datashare%'
 AND object_name LIKE '%tickit%'
 ORDER BY object_name
LIMIT 5;

share_type | share_name | object_type | object_name
-----+-----+-----+-----+
OUTBOUND | tickit_datashare | table | public.tickit_category_redshift
OUTBOUND | tickit_datashare | table | public.tickit_date_redshift
OUTBOUND | tickit_datashare | table | public.tickit_event_redshift
OUTBOUND | tickit_datashare | table | public.tickit_listing_redshift
OUTBOUND | tickit_datashare | table | public.tickit_sales_redshift
```

```
SELECT * FROM SVV_DATASHARE_OBJECTS WHERE share_name like 'sales%';

share_type | share_name | object_type | object_name | producer_account |
producer_namespace | include_new
-----+-----+-----+-----+-----+
+-----+
OUTBOUND | salesshare | schema | public | 123456789012 |
13b8833d-17c6-4f16-8fe4-1a018f5ed00d | t
OUTBOUND | salesshare | table | public.sales | 123456789012 |
13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
```

## SVV\_DEFAULT\_PRIVILEGES

Use SVV\_DEFAULT\_PRIVILEGES to view the default privileges that a user has access to in an Amazon Redshift cluster.

SVV\_DEFAULT\_PRIVILEGES is visible to all users. Superusers can see all rows. Regular users who have the ACCESS SYSTEM TABLE permission can see all the rows. Regular users can only see default privileges granted to them.

## Table columns

| Column name    | Data type | Description                                                                                                                           |
|----------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------|
| schema_name    | text      | The name of the schema.                                                                                                               |
| object_type    | text      | The object type. Possible values are RELATION, FUNCTION, or PROCEDURE.                                                                |
| owner_id       | integer   | The owner ID. Possible value is the user ID.                                                                                          |
| owner_name     | text      | The name of the owner.                                                                                                                |
| owner_type     | text      | The owner type. Possible value is user.                                                                                               |
| privilege_type | text      | The privilege type. Possible values are INSERT, SELECT, UPDATE, DELETE, RULE, REFERENCES TRIGGER, DROP, and EXECUTE.                  |
| grantee_id     | integer   | The grantee ID. Possible values are user ID, role ID, and group ID.                                                                   |
| grantee_type   | text      | The grantee type. Possible values are user, role, and public.                                                                         |
| admin_option   | boolean   | The value that indicates whether the user can grant permissions to other users and roles. It is always false for role and group type. |

## Sample query

The following example returns the output for SVV\_DEFAULT\_PRIVILEGES.

```
SELECT * from svv_default_privileges;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| schema_name | object_type | owner_id | owner_name | owner_type | privilege_type | grantee_id | grantee_name | grantee_type | admin_option |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| public | RELATION | 106 | f | u1 | user | 107 | u2 | user | false |
| public | RELATION | 106 | f | u1 | user | 107 | u2 | user | false |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

## SVV\_DISKUSAGE

Amazon Redshift creates the SVV\_DISKUSAGE system view by joining the STV\_TBL\_PERM and STV\_BLOCKLIST tables. The SVV\_DISKUSAGE view contains information about data allocation for the tables in a database.

Use aggregate queries with SVV\_DISKUSAGE, as the following examples show, to determine the number of disk blocks allocated per database, table, slice, or column. Each data block uses 1 MB. You can also use [STV\\_PARTITIONS \(p. 1591\)](#) to view summary information about disk utilization.

SVV\_DISKUSAGE is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name    | Data type     | Description                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| db_id          | integer       | Database ID.                                                                                                                                                                                                                                                                                                                                                                           |
| name           | character(72) | Table name.                                                                                                                                                                                                                                                                                                                                                                            |
| slice          | integer       | Data slice allocated to the table.                                                                                                                                                                                                                                                                                                                                                     |
| col            | integer       | Zero-based index for the column. Every table you create has three hidden columns appended to it: INSERT_XID, DELETE_XID, and ROW_ID (OID). A table with 3 user-defined columns contains 6 actual columns, and the user-defined columns are internally numbered as 0, 1, and 2. The INSERT_XID, DELETE_XID, and ROW_ID columns are numbered 3, 4, and 5, respectively, in this example. |
| tbl            | integer       | Table ID.                                                                                                                                                                                                                                                                                                                                                                              |
| blocknum       | integer       | ID for the data block.                                                                                                                                                                                                                                                                                                                                                                 |
| num_values     | integer       | Number of values contained on the block.                                                                                                                                                                                                                                                                                                                                               |
| minvalue       | bigint        | Minimum value contained on the block.                                                                                                                                                                                                                                                                                                                                                  |
| maxvalue       | bigint        | Maximum value contained on the block.                                                                                                                                                                                                                                                                                                                                                  |
| sb_pos         | integer       | Internal identifier for the position of the super block on disk.                                                                                                                                                                                                                                                                                                                       |
| pinned         | integer       | Whether or not the block is pinned into memory as part of pre-load. 0 = false; 1 = true. Default is false.                                                                                                                                                                                                                                                                             |
| on_disk        | integer       | Whether or not the block is automatically stored on disk. 0 = false; 1 = true. Default is false.                                                                                                                                                                                                                                                                                       |
| modified       | integer       | Whether or not the block has been modified. 0 = false; 1 = true. Default is false.                                                                                                                                                                                                                                                                                                     |
| hdr_modified   | integer       | Whether or not the block header has been modified. 0 = false; 1 = true. Default is false.                                                                                                                                                                                                                                                                                              |
| unsorted       | integer       | Whether or not a block is unsorted. 0 = false; 1 = true. Default is true.                                                                                                                                                                                                                                                                                                              |
| tombstone      | integer       | For internal use.                                                                                                                                                                                                                                                                                                                                                                      |
| preferred_disk | integer       | Disk number that the block should be on, unless the disk has failed. Once the disk has been fixed, the block will move back to this disk.                                                                                                                                                                                                                                              |
| temporary      | integer       | Whether or not the block contains temporary data, such as from a temporary table or intermediate query results. 0 = false; 1 = true. Default is false.                                                                                                                                                                                                                                 |
| newblock       | integer       | Indicates whether or not a block is new (true) or was never committed to disk (false). 0 = false; 1 = true.                                                                                                                                                                                                                                                                            |

## Sample queries

SVV\_DISKUSAGE contains one row per allocated disk block, so a query that selects all the rows potentially returns a very large number of rows. We recommend using only aggregate queries with SVV\_DISKUSAGE.

Return the highest number of blocks ever allocated to column 6 in the USERS table (the EMAIL column):

```
select db_id, trim(name) as tablename, max(blocknum)
from svv_diskusage
where name='users' and col=6
group by db_id, name;

db_id | tablename | max
-----+-----+-----
175857 | users | 2
(1 row)
```

The following query returns similar results for all of the columns in a large 10-column table called SALESNEW. (The last three rows, for columns 10 through 12, are for the hidden metadata columns.)

```
select db_id, trim(name) as tablename, col, tbl, max(blocknum)
from svv_diskusage
where name='salesnew'
group by db_id, name, col, tbl
order by db_id, name, col, tbl;

db_id | tablename | col | tbl | max
-----+-----+-----+-----+
175857 | salesnew | 0 | 187605 | 154
175857 | salesnew | 1 | 187605 | 154
175857 | salesnew | 2 | 187605 | 154
175857 | salesnew | 3 | 187605 | 154
175857 | salesnew | 4 | 187605 | 154
175857 | salesnew | 5 | 187605 | 79
175857 | salesnew | 6 | 187605 | 79
175857 | salesnew | 7 | 187605 | 302
175857 | salesnew | 8 | 187605 | 302
175857 | salesnew | 9 | 187605 | 302
175857 | salesnew | 10 | 187605 | 3
175857 | salesnew | 11 | 187605 | 2
175857 | salesnew | 12 | 187605 | 296
(13 rows)
```

## SVV\_EXTERNAL\_COLUMNS

Use SVV\_EXTERNAL\_COLUMNS to view details for columns in external tables. Use SVV\_EXTERNAL\_COLUMNS also for cross-database queries to view details on all columns from the table on unconnected databases that users have access to.

SVV\_EXTERNAL\_COLUMNS is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access. For more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#).

### Table columns

| Column name            | Data type | Description                                     |
|------------------------|-----------|-------------------------------------------------|
| redshift_database_name | text      | The name of the local Amazon Redshift database. |

| Column name   | Data type | Description                                                                                                                   |
|---------------|-----------|-------------------------------------------------------------------------------------------------------------------------------|
| schemaname    | text      | The name of the Amazon Redshift external schema for the external table.                                                       |
| tablename     | text      | The name of the external table.                                                                                               |
| columnname    | text      | The name of the column.                                                                                                       |
| external_type | text      | The data type of the column.                                                                                                  |
| columnnum     | integer   | The external column number, starting from 1.                                                                                  |
| part_key      | integer   | If the column is a partition key, the order of the key. If the column isn't a partition, the value is 0.                      |
| is_nullable   | text      | Defines whether a column is nullable or not. Some values are true, false, or " " empty string that represents no information. |

## SVV\_EXTERNAL\_DATABASES

Use SVV\_EXTERNAL\_DATABASES to view details for external databases.

SVV\_EXTERNAL\_DATABASES is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access. For more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#).

### Table columns

| Column name  | Data type | Description                                                                                                  |
|--------------|-----------|--------------------------------------------------------------------------------------------------------------|
| eskind       | integer   | The type of the external catalog for the database; 1 indicates a data catalog, 2 indicates a Hive metastore. |
| esoptions    | text      | Details of the catalog where the database resides.                                                           |
| databasename | text      | The name of the database in the external catalog.                                                            |
| location     | text      | The location of the database.                                                                                |
| parameters   | text      | Database parameters.                                                                                         |

## SVV\_EXTERNAL\_PARTITIONS

Use SVV\_EXTERNAL\_PARTITIONS to view details for partitions in external tables.

SVV\_EXTERNAL\_PARTITIONS is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access. For more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#).

## Table columns

| Column name       | Data type | Description                                                                                                                   |
|-------------------|-----------|-------------------------------------------------------------------------------------------------------------------------------|
| schemaname        | text      | The name of the Amazon Redshift external schema for the external table with the specified partitions.                         |
| tablename         | text      | The name of the external table.                                                                                               |
| values            | text      | Values for the partition.                                                                                                     |
| location          | text      | The location of the partition. The column size is limited to 128 characters. Longer values are truncated.                     |
| input_format      | text      | The input format.                                                                                                             |
| output_format     | text      | The output format.                                                                                                            |
| serialization_lib | text      | The serialization library.                                                                                                    |
| serde_parameters  | text      | SerDe parameters.                                                                                                             |
| compressed        | integer   | A value that indicates whether the partition is compressed; <b>1</b> indicates compressed, <b>0</b> indicates not compressed. |
| parameters        | text      | Partition properties.                                                                                                         |

## SVV\_EXTERNAL\_SCHEMAS

Use SVV\_EXTERNAL\_SCHEMAS to view information about external schemas. For more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#).

SVV\_EXTERNAL\_SCHEMAS is visible to all users.

## Table columns

| Column name  | Data type | Description                                                                                                                                                                                                                                                     |
|--------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| esoid        | oid       | External schema ID.                                                                                                                                                                                                                                             |
| eskind       | smallint  | The type of the external catalog for the external schema: 1 indicates a data catalog, 2 indicates a Hive metastore, 3 indicates a federated query to Aurora PostgreSQL or Amazon RDS PostgreSQL, and 4 indicates a schema for another Amazon Redshift database. |
| schemaname   | name      | External schema name.                                                                                                                                                                                                                                           |
| esowner      | integer   | User ID of the external schema owner.                                                                                                                                                                                                                           |
| databasename | text      | External database name.                                                                                                                                                                                                                                         |

| Column name | Data type | Description              |
|-------------|-----------|--------------------------|
| esoptions   | text      | External schema options. |

## Example

The following example shows details for external schemas.

```
select * from svv_external_schemas;
 esoid | eskind | schemaname | esowner | databasename | esoptions
-----+-----+-----+-----+-----+
+-----+
100133 | 1 | spectrum | 100 | redshift | {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

## SVV\_EXTERNAL\_TABLES

Use SVV\_EXTERNAL\_TABLES to view details for external tables; for more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#). Use SVV\_EXTERNAL\_TABLES also for cross-database queries to view metadata on all tables on unconnected databases that users have access to.

SVV\_EXTERNAL\_TABLES is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

## Table columns

| Column name            | Data type | Description                                                                                                            |
|------------------------|-----------|------------------------------------------------------------------------------------------------------------------------|
| redshift_database_name | text      | The name of the local Amazon Redshift database.                                                                        |
| schemaname             | text      | The name of the Amazon Redshift external schema for the external table.                                                |
| tablename              | text      | The name of the external table.                                                                                        |
| tabletype              | text      | The type of table. Some values are TABLE, VIEW, MATERIALIZED VIEW, or " " empty string that represents no information. |
| location               | text      | The location of the table.                                                                                             |
| input_format           | text      | The input format                                                                                                       |
| output_format          | text      | The output format.                                                                                                     |
| serialization_lib      | text      | The serialization library.                                                                                             |
| serde_parameters       | text      | SerDe parameters.                                                                                                      |
| compressed             | integer   | A value that indicates whether the table is compressed;                                                                |

| Column name | Data type | Description                                                       |
|-------------|-----------|-------------------------------------------------------------------|
|             |           | <b>1</b> indicates compressed, <b>0</b> indicates not compressed. |
| parameters  | text      | Table properties.                                                 |

## Example

The following example shows details `svv_external_tables` with a predicate on the external schema used by a federated query.

```
select schemaname, tablename from svv_external_tables where schemaname = 'apg_tpch';
schemaname | tablename
-----+-----
apg_tpch | customer
apg_tpch | lineitem
apg_tpch | nation
apg_tpch | orders
apg_tpch | part
apg_tpch | partsupp
apg_tpch | region
apg_tpch | supplier
(8 rows)
```

## SVV\_FUNCTION\_PRIVILEGES

Use `SVV_RELATION_PRIVILEGES` to view the function permissions that are explicitly granted to users, roles, and groups in the current database.

`SVV_FUNCTION_PRIVILEGES` is visible to all users. Superusers can see all rows. Regular users who have the `ACCESS SYSTEM TABLE` permission can see all rows. Regular users can only see identities to which they have access or for which they are the identities owner.

## Table columns

| Column name    | Data type | Description                                                                                                                                         |
|----------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| namespace_name | text      | The name of the namespace where a specified function exists.                                                                                        |
| function_name  | text      | The name of the function.                                                                                                                           |
| argument_types | text      | The string that represents the type of input argument for a function.                                                                               |
| privilege_type | text      | The type of the permission. Possible value is EXECUTE.                                                                                              |
| identity_id    | integer   | The ID of the identity. Possible values are user ID, role ID, or group ID.                                                                          |
| identity_name  | text      | The name of the identity.                                                                                                                           |
| identity_type  | text      | The type of the identity. Possible values are user, role, group, or public.                                                                         |
| admin_option   | boolean   | A value that indicates whether the user can grant the permission to other users and roles. It is always false for the role and group identity type. |

## Sample query

The following example displays the result of the SVV\_FUNCTION\_PRIVILEGES.

```
SELECT
 namespace_name, function_name, argument_types, privilege_type, identity_name, identity_type, admin_option
FROM svv_function_privileges
WHERE identity_name IN ('role1', 'reguser');

namespace_name | function_name | argument_types | privilege_type |
identity_name | identity_type | admin_option
-----+-----+-----+-----+-----+
public | test_func1 | integer | EXECUTE | role1
| role | False
public | test_func2 | integer, character varying | EXECUTE | reguser
| user | False
```

## SVV\_IAM\_PRIVILEGES

Use SVV\_IAM\_PRIVILEGES to view explicitly granted IAM privileges on users, roles and groups. SVV\_IAM\_PRIVILEGES is visible to all users. Superusers can see all rows. Regular users who have the ACCESS SYSTEM TABLE system privilege can see all rows. Regular users can only see entries they have access to.

### Table columns

| Column name   | Data type | Description                                                                            |
|---------------|-----------|----------------------------------------------------------------------------------------|
| iam_arn       | text      | Name of the namespace.                                                                 |
| command_type  |           | Privilege types. Possible values are COPY, UNLOAD, CREATE MODEL, or EXTERNAL FUNCTION. |
| identity_id   | integer   | Identity ID. Possible values are user ID, role ID, or group ID.                        |
| identity_name | text      | Identity name.                                                                         |
| identity_type | text      | Identity type. Possible values are user, role, group, or public.                       |

## Sample queries

The following example shows the results of SVV\_IAM\_PRIVILEGES.

```
SELECT * from SVV_IAM_PRIVILEGES ORDER BY IDENTITY_ID;
iam_arn | command_type | identity_id | identity_name | identity_type
-----+-----+-----+-----+-----+
default-aws-iam-role | COPY | 0 | public | public
default-aws-iam-role | UNLOAD | 0 | public | public
default-aws-iam-role | CREATE MODEL | 0 | public | public
default-aws-iam-role | EXFUNC | 0 | public | public
default-aws-iam-role | COPY | 106 | u1 | user
default-aws-iam-role | UNLOAD | 106 | u1 | user
default-aws-iam-role | CREATE MODEL | 106 | u1 | user
default-aws-iam-role | EXFUNC | 106 | u1 | user
```

|                      |              |        |    |      |
|----------------------|--------------|--------|----|------|
| default-aws-iam-role | COPY         | 118413 | r1 | role |
| default-aws-iam-role | UNLOAD       | 118413 | r1 | role |
| default-aws-iam-role | CREATE MODEL | 118413 | r1 | role |
| default-aws-iam-role | EXFUNC       | 118413 | r1 | role |

(12 rows)

## SVV\_INTERLEAVED\_COLUMNS

Use the SVV\_INTERLEAVED\_COLUMNS view to help determine whether a table that uses interleaved sort keys should be reindexed using [VACUUM REINDEX \(p. 1017\)](#). For more information about how to determine how often to run VACUUM and when to run a VACUUM REINDEX, see [Managing vacuum times \(p. 112\)](#).

SVV\_INTERLEAVED\_COLUMNS is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name      | Data type     | Description                                                                                                                                                                                                                                                |
|------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tbl              | integer       | Table ID.                                                                                                                                                                                                                                                  |
| col              | integer       | Zero-based index for the column.                                                                                                                                                                                                                           |
| interleaved_skew | numeric(18,2) | Value that indicates of the amount of skew present in the interleaved sort key columns for a table. A value of 1.00 indicates no skew, and larger values indicate more skew. Tables with a large skew should be reindexed with the VACUUM REINDEX command. |
| last_reindex     | timestamp     | Time when the last VACUUM REINDEX was run for the specified table. This value is NULL if a table has never been reindexed or if the underlying system log table, STL_VACUUM, has been rotated since the last reindex.                                      |

### Sample queries

To identify tables that might need to be reindexed, run the following query.

```
select tbl as tbl_id, stv_tbl_perm.name as table_name,
col, interleaved_skew, last_reindex
from svv_interleaved_columns, stv_tbl_perm
where svv_interleaved_columns.tbl = stv_tbl_perm.id
and interleaved_skew is not null;

tbl_id | table_name | col | interleaved_skew | last_reindex
-----+-----+-----+-----+
100068 | lineorder | 0 | 3.65 | 2015-04-22 22:05:45
100068 | lineorder | 1 | 2.65 | 2015-04-22 22:05:45
100072 | customer | 0 | 1.65 | 2015-04-22 22:05:45
100072 | lineorder | 1 | 1.00 | 2015-04-22 22:05:45
(4 rows)
```

## SVV\_IDENTITY\_PROVIDERS

The SVV\_IDENTITY\_PROVIDERS view returns the name and additional properties for identity providers. For more information about how to create an identity provider, see [CREATE IDENTITY PROVIDER \(p. 792\)](#).

SVV\_IDENTITY\_PROVIDERS is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type | Description                                                   |
|-------------|-----------|---------------------------------------------------------------|
| uid         | integer   | The unique ID of the registered identity provider.            |
| name        | text      | The identity provider name.                                   |
| type        | text      | The identity provider type.                                   |
| instanceid  | text      | The unique differentiator between instances of the same type. |
| namespc     | text      | The namespace prefix of the identity provider.                |
| params      | text      | The JSON object with parameters for the identity provider.    |
| enabled     | bool      | Indicates if the identity provider is enabled.                |

## Sample queries

Run a query like the following, after creating identity providers, to view their properties.

```
SELECT name, type, instanceid, namespc, params, enabled
FROM svv_identity_providers
ORDER BY 1;
```

The sample output includes param descriptions.

| name                                                                                                                                                                                                                                                                                                        | type  | instanceid                           | namespc |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|--------------------------------------|---------|
| params                                                                                                                                                                                                                                                                                                      |       |                                      |         |
| enabled                                                                                                                                                                                                                                                                                                     |       |                                      |         |
| rs5517_azure_idp                                                                                                                                                                                                                                                                                            | azure | e40d4bb2-7670-44ae-bfb8-5db013221d73 | abc     |
| {"issuer": "https://login.microsoftonline.com/e40d4bb2-7670-44ae-bfb8-5db013221d73/v2.0", "client_id": "871c010f-5e61-4fb1-83ac-98610a7e9110", "client_secret": "", "audience": ["https://analysis.windows.net/powerbi/connector/AmazonRedshift", "https://analysis.windows.net/powerbi/connector/AWSRDS"]} |       |                                      |         |
| (1 row)                                                                                                                                                                                                                                                                                                     |       |                                      |         |

## SVV\_LANGUAGE\_PRIVILEGES

Use SVV\_LANGUAGE\_PRIVILEGES to view the language permissions that are explicitly granted to users, roles, and groups in the current database.

SVV\_LANGUAGE\_PRIVILEGES is visible to all users. Superusers can see all rows. Regular users who have the ACCESS SYSTEM TABLE permission can see all rows. Regular users can only see identities to which they have access or for which they are the identities owner.

## Table columns

| Column name    | Data type | Description                                                                                                                                         |
|----------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| language_name  | text      | The name of the language.                                                                                                                           |
| privilege_type | text      | The type of the permission. Possible value is USAGE.                                                                                                |
| identity_id    | integer   | The ID of the identity. Possible values are user ID, role ID, or group ID.                                                                          |
| identity_name  | text      | The name of the identity.                                                                                                                           |
| identity_type  | text      | The type of the identity. Possible values are user, role, group, or public.                                                                         |
| admin_option   | boolean   | A value that indicates whether the user can grant the permission to other users and roles. It is always false for the role and group identity type. |

## Sample query

The following example displays the result of the SVV\_LANGUAGE\_PRIVILEGES.

```
SELECT language_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_language_privileges
WHERE identity_name IN ('role1', 'reguser');

language_name | privilege_type | identity_name | identity_type | admin_option
-----+-----+-----+-----+-----+
 exfunc | USAGE | reguser | user | False
 exfunc | USAGE | role1 | role | False
plpythonu | USAGE | reguser | user | False
```

## SVV\_MASKING\_POLICY

Use SVV\_MASKING\_POLICY to view all masking policies created on the cluster.

Only users with the [sys:secadmin](#) role can view SVV\_MASKING\_POLICY. Regular users will see 0 rows.

## Table columns

| Column name     | Data type | Description                                                            |
|-----------------|-----------|------------------------------------------------------------------------|
| policy_database | text      | The name of the database in which the masking policy was created.      |
| policy_name     | text      | The name of the masking policy.                                        |
| input_columns   | text      | The attributes provided in the WITH clause of CREATE POLICY statement. |

| Column name          | Data type | Description                                                    |
|----------------------|-----------|----------------------------------------------------------------|
| policy_expression    | text      | The masking expression used in the policy.                     |
| policy_modified_by   | text      | The name of the user who last modified the policy.             |
| policy_modified_time | timestamp | The timestamp of when the policy was created or last modified. |

## SVV\_ML\_MODEL\_INFO

State information about the current state of the machine learning model.

SVV\_ML\_MODEL\_INFO is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name    | Data type | Description                                                                                                                                                                                                                     |
|----------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| database_name  | char(128) | The database of the model.                                                                                                                                                                                                      |
| schema_name    | char(128) | The schema of the model.                                                                                                                                                                                                        |
| user_name      | char(128) | The owner of the model.                                                                                                                                                                                                         |
| model_name     | char(128) | The name of the model.                                                                                                                                                                                                          |
| life_cycle     | char(20)  | The lifecycle status of the model.                                                                                                                                                                                              |
| is_refreshable | integer   | The state of the model whether it is refreshable if original tables and columns in the training query still exist and the user still has the permissions to them. Possible values are: 1 (refreshable) and 0 (not refreshable). |
| model_state    | char(128) | The current state of the model.                                                                                                                                                                                                 |

### Sample query

The following query displays the current state of machine learning models.

```
SELECT schema_name, model_name, model_state
FROM svv_ml_model_info;

schema_name | model_name | model_state
-----+-----+-----+
public | customer_churn_auto_model | Train Model On SageMaker In Progress
public | customer_churn_xgboost_model | Model is Ready
(2 row)
```

## SVV\_ML\_MODEL\_PRIVILEGES

Use SVV\_ML\_MODEL\_PRIVILEGES to view the machine learning model permissions that are explicitly granted to users, roles, and groups in the cluster.

SVV\_ML\_MODEL\_PRIVILEGES is visible to all users. Superusers can see all rows. Regular users who have the ACCESS SYSTEM TABLE permission can see all rows. Regular users can only see identities to which they have access or for which they are the identities owner.

### Table columns

| Column name    | Data type | Description                                                                                                                                         |
|----------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| namespace_name | text      | The name of the namespace where a specified machine learning model exists.                                                                          |
| model_name     | text      | The name of the machine learning model.                                                                                                             |
| privilege_type | text      | The type of the permission. Possible value is EXECUTE.                                                                                              |
| identity_id    | integer   | The ID of the identity. Possible values are user ID, role ID, or group ID.                                                                          |
| identity_name  | text      | The name of the identity.                                                                                                                           |
| identity_type  | text      | The type of the identity. Possible values are user, role, group, or public.                                                                         |
| admin_option   | boolean   | A value that indicates whether the user can grant the permission to other users and roles. It is always false for the role and group identity type. |

### Sample query

The following example displays the result of the SVV\_ML\_MODEL\_PRIVILEGES.

```
SELECT
 namespace_name, model_name, model_version, privilege_type, identity_name, identity_type, admin_option
FROM svv_ml_model_privileges
WHERE model_name = 'test_model';

namespace_name | model_name | model_version | privilege_type | identity_name | identity_type | admin_option
-----+-----+-----+-----+-----+-----+-----+
public | test_model | 1 | EXECUTE | reguser | user
| False
public | test_model | 1 | EXECUTE | role1 | role
| False
```

## SVV\_MV\_DEPENDENCY

The SVV\_MV\_DEPENDENCY table shows the dependencies of materialized views on other materialized views within Amazon Redshift.

For more information about materialized views, see [Creating materialized views in Amazon Redshift \(p. 198\)](#).

SVV\_MV\_DEPENDENCY is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name             | Data type | Description                                                                |
|-------------------------|-----------|----------------------------------------------------------------------------|
| database_name           | char(128) | The database that contains the specified materialized view.                |
| schema_name             | char(128) | The schema of the materialized view.                                       |
| name                    | char(128) | The name of the materialized view.                                         |
| dependent_database_name | char(128) | The materialized view database on which this materialized view depends.    |
| dependent_schema_name   | char(128) | The materialized view schema on which this materialized view depends.      |
| dependent_name          | char(128) | The name of the materialized view on which this materialized view depends. |

### Sample query

The following query returns an output row that indicates that the materialized view mv\_over\_foo uses the materialized view mv\_foo in its definition as a dependency.

```
CREATE SCHEMA test_ivm_setup;
CREATE TABLE test_ivm_setup.foo(a INT);
CREATE MATERIALIZED VIEW test_ivm_setup.mv_foo AS SELECT * FROM test_ivm_setup.foo;
CREATE MATERIALIZED VIEW test_ivm_setup.mv_over_foo AS SELECT * FROM test_ivm_setup.mv_foo;

SELECT * FROM svv_mv_dependency;

database_name | schema_name | name | dependent_database_name |
dependent_schema_name | dependent_name
-----+-----+-----+-----+
dev | test_ivm_setup | mv_over_foo | dev |
test_ivm_setup | mv_foo | |
```

## SVV\_MV\_INFO

The SVV\_MV\_INFO table contains a row for every materialized view, whether the data is stale, and state information.

For more information about materialized views, see [Creating materialized views in Amazon Redshift \(p. 198\)](#).

SVV\_MV\_INFO is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name   | Data type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| database_name | char(128) | The database that contains the materialized view.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| schema_name   | char(128) | The schema of the database.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| user_name     | char(128) | The user who owns the materialized view.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| name          | char(128) | The materialized view name.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| is_stale      | char(1)   | A t indicates that the materialized view is stale. A <i>stale</i> materialized view is one where the base tables have been updated but the materialized view hasn't been refreshed. This information might not be accurate if a refresh hasn't been run since the last restart.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| state         | integer   | The state of the materialized view as follows: <ul style="list-style-type: none"> <li>• 0 – The materialized view is fully recomputed when refreshed.</li> <li>• 1 – The materialized view is incremental.</li> <li>• 101 – The materialized view can't be refreshed due to a dropped column. This constraint applies even if the column isn't used in the materialized view.</li> <li>• 102 – The materialized view can't be refreshed due to a changed column type. This constraint applies even if the column isn't used in the materialized view.</li> <li>• 103 – The materialized view can't be refreshed due to a renamed table.</li> <li>• 104 – The materialized view can't be refreshed due to a renamed column. This constraint applies even if the column isn't used in the materialized view.</li> <li>• 105 – The materialized view can't be refreshed due to a renamed schema.</li> </ul> |
| autorewrite   | char(1)   | A t indicates that the materialized view is eligible for automatic rewriting of queries.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| autorefresh   | char(1)   | A t indicates that the materialized view can be automatically refreshed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

## Sample query

To view the state of all materialized views, run the following query.

```
select * from svv_mv_info;
```

This query returns the following sample output.

| database_name | schema_name      | user_name | name   | is_stale | state |
|---------------|------------------|-----------|--------|----------|-------|
| autorefresh   | autorewrite      |           |        |          |       |
| dev 1         | test_ivm_setup 0 | catch-22  | mv     | f        | 1     |
| dev 0         | test_ivm_setup 1 | lotr      | old_mv | t        | 1     |

## SVV\_QUERY\_INFLIGHT

Use the SVV\_QUERY\_INFLIGHT view to determine what queries are currently running on the database. This view joins [STV\\_INFLIGHT \(p. 1583\)](#) to [STL\\_QUERYTEXT \(p. 1544\)](#). SVV\_QUERY\_INFLIGHT does not show leader-node only queries. For more information, see [Leader node-only functions \(p. 1020\)](#).

SVV\_QUERY\_INFLIGHT is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type      | Description                                                                                                                                                                                                                                   |
|-------------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer        | ID of user who generated entry.                                                                                                                                                                                                               |
| slice       | integer        | Slice where the query is running.                                                                                                                                                                                                             |
| query       | integer        | Query ID. Can be used to join various other system tables and views.                                                                                                                                                                          |
| pid         | integer        | Process ID. All of the queries in a session are run in the same process, so this value remains constant if you run a series of queries in the same session. You can use this column to join to the <a href="#">STL_ERROR (p. 1513)</a> table. |
| starttime   | timestamp      | Time that the query started.                                                                                                                                                                                                                  |
| suspended   | integer        | Whether the query is suspended: <b>0</b> = false; <b>1</b> = true.                                                                                                                                                                            |
| text        | character(200) | Query text, in 200-character increments.                                                                                                                                                                                                      |
| sequence    | integer        | Sequence number for segments of query statements.                                                                                                                                                                                             |

### Sample queries

The sample output below shows two queries currently running, the SVV\_QUERY\_INFLIGHT query itself and query 428, which is split into three rows in the table. (The starttime and statement columns are truncated in this sample output.)

```
select slice, query, pid, starttime, suspended, trim(text) as statement, sequence
from svv_query_inflight
order by query, sequence;
```

| slice query  pid | starttime | suspended | statement | sequence |
|------------------|-----------|-----------|-----------|----------|
|------------------|-----------|-----------|-----------|----------|

|                                                               |  |  |  |  |  |  |
|---------------------------------------------------------------|--|--|--|--|--|--|
| 1012   428   1658   2012-04-10 13:53:...   0   select ...   0 |  |  |  |  |  |  |
| 1012   428   1658   2012-04-10 13:53:...   0   enueid ...   1 |  |  |  |  |  |  |
| 1012   428   1658   2012-04-10 13:53:...   0   atname,...   2 |  |  |  |  |  |  |
| 1012   429   1608   2012-04-10 13:53:...   0   select ...   0 |  |  |  |  |  |  |
| (4 rows)                                                      |  |  |  |  |  |  |

## SVV\_QUERY\_STATE

Use SVV\_QUERY\_STATE to view information about the runtime of currently running queries.

The SVV\_QUERY\_STATE view contains a data subset of the STV\_EXEC\_STATE table.

SVV\_QUERY\_STATE is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name  | Data type        | Description                                                                                                                                                                                                                                    |
|--------------|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid       | integer          | ID of user who generated entry.                                                                                                                                                                                                                |
| query        | integer          | Query ID. Can be used to join various other system tables and views.                                                                                                                                                                           |
| seg          | integer          | Number of the query segment that is running. A query consists of multiple segments, and each segment consists of one or more steps. Query segments can run in parallel. Each segment runs in a single process.                                 |
| step         | integer          | Number of the query step that is running. A step is the smallest unit of query runtime. Each step represents a discrete unit of work, such as scanning a table, returning results, or sorting data.                                            |
| maxtime      | interval         | Maximum amount of time (in microseconds) for this step to run.                                                                                                                                                                                 |
| avgtime      | interval         | Average time (in microseconds) for this step to run.                                                                                                                                                                                           |
| rows         | bigint           | Number of rows produced by the step that is running.                                                                                                                                                                                           |
| bytes        | bigint           | Number of bytes produced by the step that is running.                                                                                                                                                                                          |
| cpu          | bigint           | For internal use.                                                                                                                                                                                                                              |
| memory       | bigint           | For internal use.                                                                                                                                                                                                                              |
| rate_row     | double precision | Rows-per-second rate since the query started, computed by summing the rows and dividing by the number of seconds from when the query started to the current time.                                                                              |
| rate_byte    | double precision | Bytes-per-second rate since the query started, computed by summing the bytes and dividing by the number of seconds from when the query started to the current time.                                                                            |
| label        | character(25)    | Query label: a name for the step, such as scan or sort.                                                                                                                                                                                        |
| is_diskbased | character(1)     | Whether this step of the query is running as a disk-based operation: true ( <b>t</b> ) or false ( <b>f</b> ). Only certain steps, such as hash, sort, and aggregate steps, can go to disk. Many types of steps are always performed in memory. |

| Column name     | Data type    | Description                                                                                                                                                                                                                                                 |
|-----------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| workmem         | bigint       | Amount of working memory (in bytes) assigned to the query step.                                                                                                                                                                                             |
| num_parts       | integer      | Number of partitions a hash table is divided into during a hash step. A positive number in this column does not imply that the hash step runs as a disk-based operation. Check the value in the IS_DISKBASED column to see if the hash step was disk-based. |
| is_rrscan       | character(1) | If true ( <b>t</b> ), indicates that range-restricted scan was used on the step. Default is false ( <b>f</b> ).                                                                                                                                             |
| is_delayed_scan | character(1) | If true ( <b>t</b> ), indicates that delayed scan was used on the step. Default is false ( <b>f</b> ).                                                                                                                                                      |

## Sample queries

### Determining the processing time of a query by step

The following query shows how long each step of the query with query ID 279 took to run and how many data rows Amazon Redshift processed:

```
select query, seg, step, maxtime, avgtime, rows, label
from svv_query_state
where query = 279
order by query, seg, step;
```

This query retrieves the processing information about query 279, as shown in the following sample output:

| query | seg | step | maxtime | avgtime | rows    | label      |
|-------|-----|------|---------|---------|---------|------------|
| 279   | 3   | 0    | 1658054 | 1645711 | 1405360 | scan       |
| 279   | 3   | 1    | 1658072 | 1645809 | 0       | project    |
| 279   | 3   | 2    | 1658074 | 1645812 | 1405434 | insert     |
| 279   | 3   | 3    | 1658080 | 1645816 | 1405437 | distribute |
| 279   | 4   | 0    | 1677443 | 1666189 | 1268431 | scan       |
| 279   | 4   | 1    | 1677446 | 1666192 | 1268434 | insert     |
| 279   | 4   | 2    | 1677451 | 1666195 | 0       | aggr       |

(7 rows)

### Determining if any active queries are currently running on disk

The following query shows if any active queries are currently running on disk:

```
select query, label, is_diskbased from svv_query_state
where is_diskbased = 't';
```

This sample output shows any active queries currently running on disk:

| query | label        | is_diskbased |
|-------|--------------|--------------|
| 1025  | hash tbl=142 | t            |

(1 row)

## SVV\_REDSHIFT\_COLUMNS

Use SVV\_REDSHIFT\_COLUMNS to view a list of all columns that a user has access to. This set of columns includes the columns on the cluster and the columns from datashares provided by remote clusters.

SVV\_REDSHIFT\_COLUMNS is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

### Table columns

| Column name      | Data type     | Description                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| database_name    | varchar(128)  | The name of the database where the table containing the columns exists.                                                                                                                                                                                                                                                                                                                                                |
| schema_name      | varchar(128)  | The name of the schema for the table.                                                                                                                                                                                                                                                                                                                                                                                  |
| table_name       | varchar(128)  | The name of the table.                                                                                                                                                                                                                                                                                                                                                                                                 |
| column_name      | varchar(128)  | The name of a column.                                                                                                                                                                                                                                                                                                                                                                                                  |
| ordinal_position | integer       | The position of the column in the table.                                                                                                                                                                                                                                                                                                                                                                               |
| data_type        | varchar(32)   | The data type of the column.                                                                                                                                                                                                                                                                                                                                                                                           |
| column_default   | varchar(4000) | The default value of the column.                                                                                                                                                                                                                                                                                                                                                                                       |
| is_nullable      | varchar(3)    | A value that defines whether a column is nullable. Possible values are yes, no, and " " (an empty string that represents no information).                                                                                                                                                                                                                                                                              |
| encoding         | varchar(128)  | The encoding type of the column.                                                                                                                                                                                                                                                                                                                                                                                       |
| distkey          | boolean       | A value that is true if this column is the distribution key for the table, and false otherwise.                                                                                                                                                                                                                                                                                                                        |
| sortkey          | integer       | A value that specifies the order of the column in the sort key.<br><br>If the table uses a compound sort key, then all columns that are part of the sort key have a positive value that indicates the position of the column in the sort key.<br><br>If the table uses an interleaved sort key, then each column that is part of the sort key has a value that is alternately positive or negative. Here, the absolute |

| Column name | Data type    | Description                                                                                                          |
|-------------|--------------|----------------------------------------------------------------------------------------------------------------------|
|             |              | value indicates the position of the column in the sort key.<br>If sortkey is 0, the column isn't part of a sort key. |
| column_acl  | varchar(128) | A string that defines the permissions for the specified user or user group for the column.                           |
| remarks     | varchar(256) | Remarks.                                                                                                             |

## Sample query

The following example returns the output of SVV\_REDSHIFT\_COLUMNS.

```
SELECT *
FROM svv_redshift_columns
WHERE database_name = 'tickit_db'
 AND TABLE_NAME = 'tickit_sales_redshift'
ORDER BY COLUMN_NAME,
 TABLE_NAME,
 database_name
LIMIT 5;
```

| database_name | schema_name | table_name            | column_name | ordinal_position | data_type | column_default | is_nullable | encoding | distkey | sortkey | column_acl | remarks |
|---------------|-------------|-----------------------|-------------|------------------|-----------|----------------|-------------|----------|---------|---------|------------|---------|
| tickit_db     | public      | tickit_sales_redshift | buyerid     | 4                | integer   | NO             | az64        |          | False   | 0       |            |         |
| tickit_db     | public      | tickit_sales_redshift | commission  | 9                | numeric   | (8,2)          | YES         | az64     | False   | 0       |            |         |
| tickit_db     | public      | tickit_sales_redshift | dateid      | 6                | smallint  | NO             | none        |          | False   | 1       |            |         |
| tickit_db     | public      | tickit_sales_redshift | eventid     | 5                | integer   | NO             | az64        |          | False   | 0       |            |         |
| tickit_db     | public      | tickit_sales_redshift | listid      | 2                | integer   | NO             | az64        |          | True    | 0       |            |         |

## SVV\_REDSHIFT\_DATABASES

Use SVV\_REDSHIFT\_DATABASES to view a list of all the databases that a user has access to. This includes the databases on the cluster and the databases created from datashares provided by remote clusters.

SVV\_REDSHIFT\_DATABASES is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

## Table columns

| Column name   | Data type    | Description               |
|---------------|--------------|---------------------------|
| database_name | varchar(128) | The name of the database. |

| Column name      | Data type    | Description                                                                                    |
|------------------|--------------|------------------------------------------------------------------------------------------------|
| database_owner   | integer      | The database owner user ID.                                                                    |
| database_type    | varchar(16)  | The type of database. Possible types are local or shared databases.                            |
| database_acl     | varchar(256) | The string that defines the permissions for the specified user or user group for the database. |
| database_options | text         | The properties of the database.                                                                |

## Sample query

The following example returns the output for SVV\_REDSHIFT\_DATABASES.

```
SELECT database_name, database_type
FROM svv_redshift_databases
WHERE database_name = 'tickit_db';

database_name | database_owner | database_type | database_acl | database_options
-----+-----+-----+-----+-----+
 tickit_db | shared | | | |
```

## SVV\_REDSHIFT\_FUNCTIONS

Use SVV\_REDSHIFT\_FUNCTIONS to view a list of all functions that a user has access to. This set of functions includes the functions on the cluster and the functions from datashares provided by remote clusters.

### Table columns

| Column name   | Data type    | Description                                                                                              |
|---------------|--------------|----------------------------------------------------------------------------------------------------------|
| database_name | varchar(128) | The name of the database where the cluster that has these functions exists.                              |
| schema_name   | varchar(128) | The name of the schema that specifies a given function.                                                  |
| function_name | varchar(128) | The name of a specified function.                                                                        |
| function_type | varchar(128) | The type of function. Possible values are regular functions, aggregate functions, and stored procedures. |
| argument_type | varchar(512) | A string that represents the type of a function's input argument.                                        |

| Column name | Data type    | Description                                 |
|-------------|--------------|---------------------------------------------|
| result_type | varchar(128) | The data type of a function's return value. |

## Sample query

The following example returns the output of SVV\_REDSHIFT\_FUNCTIONS.

```
SELECT *
FROM svv_redshift_functions
WHERE database_name = 'tickit_db'
 AND SCHEMA_NAME = 'public'
ORDER BY function_name
LIMIT 5;

database_name | schema_name | function_name | function_type | argument_type |
result_type
-----+-----+-----+-----+-----+
 +-----+-----+-----+-----+
 tickit_db | public | shared_function | REGULAR FUNCTION | integer, integer |
 integer
```

## SVV\_REDSHIFT\_SCHEMAS

Use SVV\_REDSHIFT\_SCHEMAS to view a list of all schemas that a user has access to. This set of schemas includes the schemas on the cluster and the schemas from datashares provided by remote clusters.

## Table columns

| Column name   | Data type    | Description                                                                                  |
|---------------|--------------|----------------------------------------------------------------------------------------------|
| database_name | varchar(128) | The name of the database where a specified schema exists.                                    |
| schema_name   | varchar(128) | The namespace or schema name.                                                                |
| schema_owner  | integer      | The internal user ID of the schema owner.                                                    |
| schema_type   | varchar(16)  | The type of the schema. Possible values are shared and local schemas.                        |
| schema_acl    | varchar(128) | The string that defines the permissions for the specified user or user group for the schema. |
| schema_option | varchar(128) | The options of the schema.                                                                   |

## Sample query

The following example returns the output of SVV\_REDSHIFT\_SCHEMAS.

```
SELECT *
FROM svv_redshift_schemas
WHERE database_name = 'tickit_db'
ORDER BY database_name,
SCHEMA_NAME;

database_name | schema_name | schema_owner | schema_type | schema_acl |
 | schema_option |
-----+-----+-----+-----+-----+
tickit_db | public | 1 | shared |
```

## SVV\_REDSHIFT\_TABLES

Use SVV\_REDSHIFT\_TABLES to view a list of all tables that a user has access to. This set of tables includes the tables on the cluster and the tables from datashares provided by remote clusters.

### Table columns

| Column name   | Data type    | Description                                                                                 |
|---------------|--------------|---------------------------------------------------------------------------------------------|
| database_name | varchar(128) | The name of the database where a specified table exists.                                    |
| schema_name   | varchar(128) | The name the schema for the table.                                                          |
| table_name    | varchar(128) | The name of the table.                                                                      |
| table_type    | varchar(128) | The type of table. Possible values are views and tables.                                    |
| table_acl     | varchar(128) | The string that defines the permissions for the specified user or user group for the table. |
| remarks       | varchar(128) | Remarks.                                                                                    |

### Sample query

The following example returns the output of SVV\_REDSHIFT\_TABLES.

```
SELECT *
FROM svv_redshift_tables
WHERE database_name = 'tickit_db' AND TABLE_NAME LIKE 'tickit_%'
ORDER BY database_name,
TABLE_NAME;

database_name | schema_name | table_name | table_type | table_acl | remarks
-----+-----+-----+-----+-----+-----+
tickit_db | public | tickit_category_redshift | TABLE | | |
tickit_db | public | tickit_date_redshift | TABLE | | |
tickit_db | public | tickit_event_redshift | TABLE | | |
tickit_db | public | tickit_listing_redshift | TABLE | | |
tickit_db | public | tickit_sales_redshift | TABLE | | |
tickit_db | public | tickit_users_redshift | TABLE | | |
tickit_db | public | tickit_venue_redshift | TABLE | | |
```

## SVV\_RELATION\_PRIVILEGES

Use SVV\_RELATION\_PRIVILEGES to view the relation (tables and views) permissions that are explicitly granted to users, roles, and groups in the current database.

SVV\_RELATION\_PRIVILEGES is visible to all users. Superusers can see all rows. Regular users who have the SYSLOG ACCESS permission can see all rows. Regular users can only see identities to which they have access or for which they are the identities' owner. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name    | Data type | Description                                                                                                                                         |
|----------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| namespace_name | text      | The name of the namespace where a specified relation exists.                                                                                        |
| relation_name  | text      | The name of the relation.                                                                                                                           |
| privilege_type | text      | The type of the permission. Possible values are INSERT, SELECT, UPDATE, DELETE, REFERENCES, or DROP.                                                |
| identity_id    | integer   | The ID of the identity. Possible values are user ID, role ID, or group ID.                                                                          |
| identity_name  | text      | The name of the identity.                                                                                                                           |
| identity_type  | text      | The type of the identity. Possible values are user, role, group, or public.                                                                         |
| admin_option   | boolean   | A value that indicates whether the user can grant the permission to other users and roles. It is always false for the role and group identity type. |

### Sample query

The following example displays the result of the SVV\_RELATION\_PRIVILEGES.

```
SELECT namespace_name,relation_name,privilege_type,identity_name,identity_type,admin_option
 FROM svv_relation_privileges
 WHERE relation_name = 'orders' AND privilege_type = 'SELECT';

+-----+-----+-----+-----+-----+
| namespace_name | relation_name | privilege_type | identity_name | identity_type | admin_option |
+-----+-----+-----+-----+-----+
public	orders	SELECT	reguser	user	False
False					
public	orders	SELECT	role1	role	False
False					
```

## SVV\_RLS\_APPLIED\_POLICY

Use SVV\_RLS\_APPLIED\_POLICY to trace the application of RLS policies on queries that reference RLS-protected relations.

Only superuser, sys:operator, or users with system privilege ACCESS SYSTEM TABLE can query this view. Note that sys:secadmin isn't granted this system permission.

## Table columns

| Column name | Data type | Description                                                                                                                                                                                                                     |
|-------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| username    | text      | The name of the user that ran the query.                                                                                                                                                                                        |
| query       | integer   | The ID of the query.                                                                                                                                                                                                            |
| xid         | long      | The context of the transaction.                                                                                                                                                                                                 |
| pid         | integer   | The leader process running the query.                                                                                                                                                                                           |
| recordtime  | time      | The time when the query was recorded.                                                                                                                                                                                           |
| command     | char(1)   | The command for which the RLS policy was applied. Possible values are k for unknown, s for select, u for update, i for insert, y for utility, and d for delete.                                                                 |
| datname     | text      | The name of the database of the relation to which the row-level security policy is attached.                                                                                                                                    |
| relschema   | text      | The name of the schema of the relation to which the row-level security policy is attached.                                                                                                                                      |
| relname     | text      | The name of the relation to which the row-level security policy is attached.                                                                                                                                                    |
| polname     | text      | The name of the row-level security policy that is attached to the relation.                                                                                                                                                     |
| poldefault  | char(1)   | The default setting of the row-level security policy that is attached to the relation. Possible values are f for false if the default false policy has been applied and t for true if the default true policy has been applied. |

## Sample query

The following example displays the result of the SVV\_RLS\_APPLIED\_POLICY. To query the SVV\_RLS\_APPLIED\_POLICY, you must have the ACCESS SYSTEM TABLE permission.

```
-- Check what RLS policies were applied to the run query.
SELECT username, command, datname, relschema, relname, polname, poldefault
FROM svv_rls_applied_policy
WHERE datname = CURRENT_DATABASE() AND query = PG_LAST_QUERY_ID();

username | command | datname | relschema | relname | polname |
poldefault
-----+-----+-----+-----+-----+-----+
+-----+
molly | s | tickit_db | public | tickit_category_redshift | policy_concerts |
```

## SVV\_RLS\_ATTACHED\_POLICY

Use SVV\_RLS\_ATTACHED\_POLICY to view a list of all relations and users that have one or more row-level security policies attached on the currently connected database.

Only users with the sys:secadmin role can query this view.

## Table columns

| Column name | Data type | Description                                                                                                                          |
|-------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------|
| relschema   | text      | The name of the schema of the relation to which the row-level security policy is attached.                                           |
| relname     | text      | The name of the relation to which the row-level security policy is attached.                                                         |
| relkind     | text      | The type of the object, such as table.                                                                                               |
| polname     | text      | The name of the row-level security policy that is attached to the relation.                                                          |
| grantor     | text      | The name of the user that has attached this policy.                                                                                  |
| grantee     | text      | The name of the user or role that this policy has been attached to.                                                                  |
| granteekind | text      | The type of grantee. Possible values are user or role.                                                                               |
| is_pol_on   | boolean   | The parameter that indicates whether a row-level security policy is turned on or off on a table. Possible values are true and false. |
| is_rls_on   | boolean   | The parameter that indicates whether a row-level security is turned on or off on a table. Possible values are true and false.        |

## Sample query

The following example displays the result of the SVV\_RLS\_ATTACHED\_POLICY.

```
--Inspect the policy in SVV_RLS_ATTACHED_POLICY
SELECT * FROM svv_rls_attached_policy;

 relschema | relname | relkind | polname | grantor | grantee |
 granteekind | is_pol_on | is_rls_on |
-----+-----+-----+-----+-----+-----+
public | ticket_category_redshift | table | policy_concerts | bob | analyst |
role | True | True |
public | ticket_category_redshift | table | policy_concerts | bob | dbadmin |
role | True | True |
```

## SVV\_RLS\_POLICY

Use SVV\_RLS\_POLICY to view a list of all row-level security policies created on the Amazon Redshift cluster.

Only users with the sys:secadmin role can query this view.

## Table columns

| Column name | Data type | Description                                                                 |
|-------------|-----------|-----------------------------------------------------------------------------|
| poldb       | text      | The name of the database in which the row-level security policy is created. |

| Column name     | Data type | Description                                                                       |
|-----------------|-----------|-----------------------------------------------------------------------------------|
| polname         | text      | The name of the row-level security policy.                                        |
| polalias        | text      | The table alias used in the policy definition.                                    |
| polatts         | text      | The attributes provided to the policy definition.                                 |
| polqual         | text      | The policy condition provided in the USING clause of the CREATE POLICY statement. |
| polenabled      | boolean   | Whether the policy is turned on globally.                                         |
| polmodifiedby   | text      | The name of the user that created or modified the policy most recently.           |
| polmodifiedtime | timestamp | The timestamp of when the policy is created or last modified.                     |

## Sample query

The following example displays the result of the SVV\_RLS\_POLICY.

```
-- Create some policies.
CREATE RLS POLICY pol1 WITH (a int) AS t USING (t.a IS NOT NULL);
CREATE RLS POLICY pol2 WITH (c varchar(10)) AS t USING (c LIKE '%public%');

-- Inspect the policy in SVV_RLS_POLICY
SELECT * FROM svv_rls_policy;

+-----+-----+-----+-----+-----+
| poldb | polname | polalias | polatts | polenabled | polmodifiedby | polmodifiedtime |
+-----+-----+-----+-----+-----+
| my_db | pol1 | t | [{"colname": "a", "type": "integer"}] | t | policy_admin | 2022-02-11 14:40:49
| my_db | pol2 | t | [{"colname": "c", "type": "character varying(10)"}] | t | policy_admin | 2022-02-11 14:41:28
+-----+-----+-----+-----+-----+
```

## SVV\_RLS\_RELATION

Use SVV\_RLS\_RELATION to view a list of all relations that are RLS-protected.

Only users with the sys:secadmin role can query this view.

### Table columns

| Column name | Data type | Description                                                         |
|-------------|-----------|---------------------------------------------------------------------|
| datname     | text      | The name of the database containing the relation.                   |
| relschema   | text      | The name of the schema containing the relation.                     |
| relname     | text      | The name of the relation.                                           |
| relkind     | text      | The type of the relation, such as tables or views.                  |
| is_rls_on   | boolean   | The parameter that indicates whether the relation is RLS-protected. |

| Column name      | Data type | Description                                                                         |
|------------------|-----------|-------------------------------------------------------------------------------------|
| is_rls_datashare | boolean   | The parameter that indicates whether the relation is RLS-protected over datashares. |

## Sample query

The following example displays the result of the SVV\_RLS\_RELATION.

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON FOR DATASHARES;

--Inspect RLS state on the relations using SVV_RLS_RELATION.
SELECT datname, relschema, relname, relkind, is_rls_on, is_rls_datashare_on FROM
svv_rls_relation ORDER BY relname;

+-----+-----+-----+-----+-----+
| datname | relschema | relname | relkind | is_rls_on |
| is_rls_datashare_on |
+-----+-----+-----+-----+-----+
| tickit_db | public | tickit_category_redshift | table | t | t
(1 row)
```

## SVV\_ROLES

Use SVV\_ROLES to view a list of roles that a user has access to.

SVV\_ROLES is visible to all users. Superusers can see all rows. Regular users who have the ACCESS SYSTEM TABLE permission can see all rows. Regular users can only see identities to which they have access or for which they are the identities owner.

## Table columns

| Column name | Data type | Description                                                             |
|-------------|-----------|-------------------------------------------------------------------------|
| role_id     | integer   | The role ID.                                                            |
| role_name   | text      | The name of the role.                                                   |
| role_owner  | text      | The name of the role owner.                                             |
| external_id | text      | The unique identifier of the role in the third-party identity provider. |

## Sample query

The following example returns the output of SVV\_ROLES.

```
SELECT role_name,role_owner FROM svv_roles WHERE role_name IN ('role1', 'role2');

+-----+-----+
| role_name | role_owner |
+-----+-----+
| role1 | superuser |
```

|                   |
|-------------------|
| role2   superuser |
|-------------------|

## SVV\_ROLE\_GRANTS

Use SVV\_ROLE\_GRANTS to view a list of roles that are explicitly granted roles in the cluster.

SVV\_ROLE\_GRANTS is visible to all users. Superusers can see all rows. Regular user who has the ACCESS SYSTEM TABLE permission can see all rows. Regular users can only see roles to which they have access or if they are the role owner.

### Table columns

| Column name       | Data type | Description                    |
|-------------------|-----------|--------------------------------|
| role_id           | integer   | The ID of the role.            |
| role_name         | text      | The name of the role.          |
| granted_role_id   | integer   | The ID for the granted role.   |
| granted_role_name | text      | The name for the granted role. |

### Sample query

The following example returns the output of SVV\_ROLE\_GRANTS.

```
GRANT ROLE role1 TO ROLE role2;
GRANT ROLE role2 TO ROLE role3;

SELECT role_name, granted_role_name FROM svv_role_grants;

role_name | granted_role_name
-----+-----
role2 | role1
role3 | role2
(2 rows)
```

## SVV\_SCHEMA\_QUOTA\_STATE

Displays the quota and the current disk usage for each schema.

Regular users can see information for schemas for which they have USAGE permission. Superusers can see information for all schemas in the current database.

SVV\_SCHEMA\_QUOTA\_STATE is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                 |
|-------------|-----------|-----------------------------|
| schema_id   | integer   | The namespace or schema ID. |

| Column name    | Data type        | Description                                                                                 |
|----------------|------------------|---------------------------------------------------------------------------------------------|
| schema_name    | character (128)  | The namespace or schema name.                                                               |
| schema_owner   | integer          | The internal user ID of the schema owner.                                                   |
| quota          | integer          | The amount of disk space (in MB) that the schema can use.                                   |
| disk_usage     | integer          | The disk space (in MB) that is currently used by the schema.                                |
| disk_usage_pct | double precision | The disk space percentage that is currently used by the schema out of the configured quota. |

## Sample query

The following example displays the quota and the current disk usage for the schema.

```
SELECT TRIM(SCHEMA_NAME) "schema_name", QUOTA, disk_usage, disk_usage_pct FROM
 svv_schema_quota_state
 WHERE SCHEMA_NAME = 'sales_schema';
schema_name | quota | disk_usage | disk_usage_pct
-----+-----+-----+-----
sales_schema | 2048 | 30 | 1.46
(1 row)
```

## SVV\_SCHEMA\_PRIVILEGES

Use SVV\_SCHEMA\_PRIVILEGES to view the schema permissions that are explicitly granted to users, roles, and groups in the current database.

SVV\_SCHEMA\_PRIVILEGES is visible to all users. Superusers can see all rows. Regular user who has the ACCESS SYSTEM TABLE permission can see all rows. Regular users can only see identities to which they have access or for which they are the identities owner.

## Table columns

| Column name    | Data type | Description                                                                 |
|----------------|-----------|-----------------------------------------------------------------------------|
| namespace_name | text      | The name of the namespace where a specified schema exists.                  |
| privilege_type | text      | The type of the permission. Possible values are USAGE or CREATE.            |
| identity_id    | integer   | The ID of the identity. Possible values are user ID, role ID, or group ID.  |
| identity_name  | text      | The name of the identity.                                                   |
| identity_type  | text      | The type of the identity. Possible values are user, role, group, or public. |

| Column name  | Data type | Description                                                                                                                                         |
|--------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| admin_option | boolean   | A value that indicates whether the user can grant the permission to other users and roles. It is always false for the role and group identity type. |

## Sample query

The following example displays the result of the SVV\_SCHEMA\_PRIVILEGES.

```
SELECT namespace_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_schema_privileges
WHERE namespace_name = 'test_schema1';

namespace_name | privilege_type | identity_name | identity_type | admin_option
-----+-----+-----+-----+-----+
test_schema1 | USAGE | reguser | user | False
test_schema1 | USAGE | role1 | role | False
```

## SVV\_SYSTEM\_PRIVILEGES

SVV\_SYSTEM\_PRIVILEGES is visible to all users. Superusers can see all rows. Regular user who has the ACCESS SYSTEM TABLE permission can see all rows. Regular users can only see identities to which they have access or for which they are the identities owner.

## Table columns

| Column name      | Data type | Description                                                     |
|------------------|-----------|-----------------------------------------------------------------|
| system_privilege | text      | The name of the system permission.                              |
| identity_id      | integer   | The ID of the identity. Possible values are user ID or role ID. |
| identity_name    | text      | The name of the identity.                                       |
| identity_type    | text      | The type of the identity. Possible values are user or role.     |

## Sample query

The following example displays the result for the specified parameters.

```
SELECT system_privilege,identity_name,identity_type FROM svv_system_privileges
WHERE system_privilege = 'ALTER TABLE' AND identity_name = 'sys:superuser';

system_privilege | identity_name | identity_type
-----+-----+-----+
ALTER TABLE | sys:superuser | role
```

## SVV\_TABLES

Use SVV\_TABLES to view tables in local and external catalogs.

SVV\_TABLES is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

## Table columns

| Column name   | Data type | Description                                                                     |
|---------------|-----------|---------------------------------------------------------------------------------|
| table_catalog | text      | The name of the catalog where the table exists.                                 |
| table_schema  | text      | The name the schema for the table.                                              |
| table_name    | text      | The name of the table.                                                          |
| table_type    | text      | The type of table. Possible values are views, external tables, and base tables. |
| remarks       | text      | Remarks.                                                                        |

## SVV\_TABLE\_INFO

Shows summary information for tables in the database. The view filters system tables and shows only user-defined tables.

You can use the SVV\_TABLE\_INFO view to diagnose and address table design issues that can influence query performance. This includes issues with compression encoding, distribution keys, sort style, data distribution skew, table size, and statistics. The SVV\_TABLE\_INFO view doesn't return any information for empty tables.

The SVV\_TABLE\_INFO view summarizes information from the [STV\\_BLOCKLIST \(p. 1579\)](#), [STV\\_NODE\\_STORAGE\\_CAPACITY \(p. 1590\)](#), [STV\\_TBL\\_PERM \(p. 1599\)](#), and [STV\\_SLICES \(p. 1598\)](#) system tables and from the [PG\\_DATABASE](#), [PG\\_ATTRIBUTE](#), [PG\\_CLASS](#), [PG\\_NAMESPACE](#), and [PG\\_TYPE](#) catalog tables.

SVV\_TABLE\_INFO is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#). To permit a user to query the view, grant SELECT permission on SVV\_TABLE\_INFO to the user.

## Table columns

| Column name | Data type | Description                                                               |
|-------------|-----------|---------------------------------------------------------------------------|
| database    | text      | Database name.                                                            |
| schema      | text      | Schema name.                                                              |
| table_id    | oid       | Table ID.                                                                 |
| table       | text      | Table name.                                                               |
| encoded     | text      | Value that indicates whether any column has compression encoding defined. |

| Column name   | Data type     | Description                                                                                                                                                                                    |
|---------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| diststyle     | text          | Distribution style or distribution key column, if key distribution is defined. Possible values include EVEN, KEY( <i>column</i> ), ALL, AUTO(ALL), AUTO(EVEN), and AUTO(KEY( <i>column</i> )). |
| sortkey1      | text          | First column in the sort key, if a sort key is defined. Possible values include <i>column</i> , AUTO(SORTKEY), and AUTO(SORTKEY( <i>column</i> )).                                             |
| max_varchar   | integer       | Size of the largest column that uses a VARCHAR data type.                                                                                                                                      |
| sortkey1_enc  | character(32) | Compression encoding of the first column in the sort key, if a sort key is defined.                                                                                                            |
| sortkey_num   | integer       | Number of columns defined as sort keys.                                                                                                                                                        |
| size          | bigint        | Size of the table, in 1-MB data blocks.                                                                                                                                                        |
| pct_used      | numeric(10,4) | Percent of available space that is used by the table.                                                                                                                                          |
| empty         | bigint        | For internal use. This column is no longer used and will be removed in a future release.                                                                                                       |
| unsorted      | numeric(5,2)  | Percent of unsorted rows in the table.                                                                                                                                                         |
| stats_off     | numeric(5,2)  | Number that indicates how stale the table's statistics are; 0 is current, 100 is out of date.                                                                                                  |
| tbl_rows      | numeric(38,0) | Total number of rows in the table. This value includes rows marked for deletion, but not yet vacuumed.                                                                                         |
| skew_sortkey1 | numeric(19,2) | Ratio of the size of the largest non-sort key column to the size of the first column of the sort key, if a sort key is defined. Use this value to evaluate the effectiveness of the sort key.  |
| skew_rows     | numeric(19,2) | Ratio of the number of rows in the slice with the most rows to the number of rows in the slice with the fewest rows.                                                                           |

| Column name            | Data type                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------------------|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| estimated_visible_rows | numeric(38,0)               | The estimated rows in the table. This value does not include rows marked for deletion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| risk_event             | text                        | <p>Risk information about a table. The field is separated into parts:</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <i>risk_type xid timestamp</i> </div> <ul style="list-style-type: none"> <li>• The <code>risk_type</code>, where 1 indicates that a COPY command with the EXPLICIT_IDS option ran. Amazon Redshift no longer checks the uniqueness of IDENTITY columns in the table. For more information, see <a href="#">EXPLICIT_IDS (p. 709)</a>.</li> <li>• The transaction ID, <code>xid</code>, that introduced the risk.</li> <li>• The timestamp when the COPY command ran.</li> </ul> <p>The following example shows the values in the field.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> 1 1107 2019-06-22<br/>07:16:11.292952 </div> |
| vacuum_sort_benefit    | numeric(12,2)               | The estimated maximum percentage improvement of scan query performance when you run vacuum sort.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| create_time            | timestamp without time zone | The timestamp for when the table was created.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

## Sample queries

The following example shows encoding, distribution style, sorting, and data skew for all user-defined tables in the database. Here, "table" must be enclosed in double quotation marks because it is a reserved word.

```
select "table", encoded, diststyle, sortkey1, skew_sortkey1, skew_rows
from svv_table_info
order by 1;

table	encoded	diststyle	sortkey1	skew_sortkey1	skew_rows
category	N	EVEN			
date	N	ALL	dateid	1.00	
event	Y	KEY(eventid)	dateid	1.00	1.02
listing	Y	KEY(listid)	dateid	1.00	1.01


```

|          |   |             |         |      |      |
|----------|---|-------------|---------|------|------|
| sales    | Y | KEY(listid) | dateid  | 1.00 | 1.02 |
| users    | Y | KEY(userid) | userid  | 1.00 | 1.01 |
| venue    | N | ALL         | venueid | 1.00 |      |
| (7 rows) |   |             |         |      |      |

## SVV\_TRANSACTIONS

Records information about transactions that currently hold locks on tables in the database. Use the SVV\_TRANSACTIONS view to identify open transactions and lock contention issues. For more information about locks, see [Managing concurrent write operations \(p. 118\)](#) and [LOCK \(p. 908\)](#).

All rows in SVV\_TRANSACTIONS, including rows generated by another user, are visible to all users.

### Table columns

| Column name          | Data type | Description                                                                                                                                                      |
|----------------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| txn_owner            | text      | Name of the owner of the transaction.                                                                                                                            |
| txn_db               | text      | Name of the database associated with the transaction.                                                                                                            |
| xid                  | bigint    | Transaction ID.                                                                                                                                                  |
| pid                  | integer   | Process ID associated with the lock.                                                                                                                             |
| txn_start            | timestamp | Start time of the transaction.                                                                                                                                   |
| lock_mode            | text      | Name of the lock mode held or requested by this process. If lock_mode is ExclusiveLock and granted is true (t), then this transaction ID is an open transaction. |
| lockable_object_type | text      | Type of object requesting or holding the lock, either relation if it is a table or transactionid if it is a transaction.                                         |
| relation             | integer   | Table ID for the table (relation) acquiring the lock. This value is NULL if lockable_object_type is transactionid.                                               |
| granted              | boolean   | Value that indicates whether that the lock has been granted (t) or is pending (f).                                                                               |

### Sample queries

The following command shows all active transactions and the locks requested by each transaction.

```
select * from svv_transactions;

 txn_
 lockable_
owner | txn_db | xid | pid | txn_start | lock_mode |
object_type | relation | granted
-----+-----+-----+-----+-----+-----+
root | dev | 438484 | 22223 | 2016-03-02 18:42:18.862254 | AccessShareLock |
relation | 100068 | t
root | dev | 438484 | 22223 | 2016-03-02 18:42:18.862254 | ExclusiveLock |
transactionid | | t
root | tickit | 438490 | 22277 | 2016-03-02 18:42:48.084037 | AccessShareLock |
relation | 50860 | t
root | tickit | 438490 | 22277 | 2016-03-02 18:42:48.084037 | AccessShareLock |
relation | 52310 | t
root | tickit | 438490 | 22277 | 2016-03-02 18:42:48.084037 | ExclusiveLock |
transactionid | | t
root | dev | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
relation | 100068 | f
root | dev | 438505 | 22378 | 2016-03-02 18:43:27.611292 | RowExclusiveLock |
relation | 16688 | t
root | dev | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessShareLock |
relation | 100064 | t
root | dev | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
relation | 100166 | t
root | dev | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
relation | 100171 | t
root | dev | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
relation | 100190 | t
root | dev | 438505 | 22378 | 2016-03-02 18:43:27.611292 | ExclusiveLock |
transactionid | | t
(12 rows)

(12 rows)
```

## SVV\_USER\_INFO

You can retrieve data about Amazon Redshift database users with the SVV\_USER\_INFO view.

### Table columns

| Column name      | Data type | Description                                                                                                                                                                                                    |
|------------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| user_name        | text      | The user name for the role.                                                                                                                                                                                    |
| user_id          | integer   | The user ID for the user.                                                                                                                                                                                      |
| createdb         | boolean   | A value that indicates whether the user has permissions to create databases.                                                                                                                                   |
| superuser        | boolean   | A value that indicates whether the user is a superuser.                                                                                                                                                        |
| catalog_update   | boolean   | A value that indicates whether the user can update system catalogs.                                                                                                                                            |
| connection_limit | text      | The number of connections that the user can open.                                                                                                                                                              |
| syslog_access    | text      | A value that indicates whether the user has access to the system logs. The two possible values are RESTRICTED and UNRESTRICTED. RESTRICTED means that users that are not superusers can see their own records. |

| Column name     | Data type | Description                                                                                                                                                                                                                                                                                                |
|-----------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                 |           | UNRESTRICTED means that user that are not superusers can see all records in the system views and tables to which they have SELECT privileges.                                                                                                                                                              |
| last_ddl_time   | timestamp | The timestamp for the last data definition language (DDL) create statement run by the user.                                                                                                                                                                                                                |
| session_timeout | integer   | The maximum time in seconds that a session remains inactive or idle before timing out. 0 indicates that no timeout is set. For information about the cluster's idle or inactive timeout setting, see <a href="#">Quotas and limits in Amazon Redshift</a> in the <i>Amazon Redshift Management Guide</i> . |
| external_user   | text      | Unique identifier of the user in the third-party identity provider.                                                                                                                                                                                                                                        |

## Sample queries

The following command retrieves user information from SVV\_USER\_INFO.

```
SELECT * FROM SVV_USER_INFO;
```

## SVV\_USER\_GRANTS

Use SVV\_USER\_GRANTS to view the list of users that are explicitly granted roles in the cluster.

SVV\_USER\_GRANTS is visible to all users. Superusers can see all rows. Regular users who have the ACCESS SYSTEM TABLE permission can see all rows. Regular users can see only current users' explicitly granted roles.

## Table columns

| Column name  | Data type | Description                                                                          |
|--------------|-----------|--------------------------------------------------------------------------------------|
| user_id      | integer   | The user ID for the user.                                                            |
| user_name    | text      | The name of the user.                                                                |
| role_id      | integer   | The role ID for the granted role.                                                    |
| role_name    | text      | The role name for the granted role.                                                  |
| admin_option | boolean   | A value that indicates whether the user can grant the role to other users and roles. |

## Sample queries

The following queries grant roles to users and show the list of users that are explicitly granted roles.

```
GRANT ROLE role1 TO reguser;
GRANT ROLE role2 TO reguser;
GRANT ROLE role1 TO superuser;
GRANT ROLE role2 TO superuser;
```

```
SELECT user_name,role_name,admin_option FROM svv_user_grants;

user_name | role_name | admin_option
-----+-----+-----
superuser | role1 | False
reguser | role1 | False
superuser | role2 | False
reguser | role2 | False
```

## SVV\_VACUUM\_PROGRESS

This view returns an estimate of how much time it will take to complete a vacuum operation that is currently in progress.

SVV\_VACUUM\_PROGRESS is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

For information about SVV\_VACUUM\_SUMMARY, see [SVV\\_VACUUM\\_SUMMARY \(p. 1464\)](#).

For information about SVL\_VACUUM\_PERCENTAGE, see [SVL\\_VACUUM\\_PERCENTAGE \(p. 1670\)](#).

### Table columns

| Column name             | Data type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| table_name              | text      | Name of the table currently being vacuumed, or the table that was last vacuumed if no operation is in progress.                                                                                                                                                                                                                                                                                                                                           |
| status                  | text      | Description of the current activity being done as part of the vacuum operation: <ul style="list-style-type: none"> <li>• Initialize</li> <li>• Sort</li> <li>• Merge</li> <li>• Delete</li> <li>• Select</li> <li>• Failed</li> <li>• Complete</li> <li>• Skipped</li> <li>• Building INTERLEAVED SORTKEY order</li> </ul>                                                                                                                                |
| time_remaining_estimate | text      | Estimated time left for the current vacuum operation to complete, in minutes and seconds: <b>5m 10s</b> , for example. An estimated time is not returned until the vacuum completes its first sort operation. If no vacuum is in progress, the last vacuum that was performed is displayed with <b>Completed</b> in the STATUS column and an empty TIME_REMAINING_ESTIMATE column. The estimate typically becomes more accurate as the vacuum progresses. |

### Sample queries

The following queries, run a few minutes apart, show that a large table named SALESNEW is being vacuumed.

```
select * from svv_vacuum_progress;

table_name	status	time_remaining_estimate
salesnew	Vacuum: initialize salesnew	

(1 row)

...
select * from svv_vacuum_progress;

table_name	status	time_remaining_estimate
salesnew	Vacuum salesnew sort	33m 21s

(1 row)
```

The following query shows that no vacuum operation is currently in progress. The last table to be vacuumed was the SALES table.

```
select * from svv_vacuum_progress;

table_name	status	time_remaining_estimate
sales	Complete	

(1 row)
```

## SVV\_VACUUM\_SUMMARY

The SVV\_VACUUM\_SUMMARY view joins the STL\_VACUUM, STL\_QUERY, and STV\_TBL\_PERM tables to summarize information about vacuum operations logged by the system. The view returns one row per table per vacuum transaction. The view records the elapsed time of the operation, the number of sort partitions created, the number of merge increments required, and deltas in row and block counts before and after the operation was performed.

SVV\_VACUUM\_SUMMARY is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

For information about SVV\_VACUUM\_PROGRESS, see [SVV\\_VACUUM\\_PROGRESS \(p. 1463\)](#).

For information about SVL\_VACUUM\_PERCENTAGE, see [SVL\\_VACUUM\\_PERCENTAGE \(p. 1670\)](#).

### Table columns

| Column name      | Data type | Description                                                                              |
|------------------|-----------|------------------------------------------------------------------------------------------|
| table_name       | text      | Name of the vacuumed table.                                                              |
| xid              | bigint    | Transaction ID of the VACUUM operation.                                                  |
| sort_partitions  | bigint    | Number of sorted partitions created during the sort phase of the vacuum operation.       |
| merge_increments | bigint    | Number of merge increments required to complete the merge phase of the vacuum operation. |
| elapsed_time     | bigint    | Elapsed runtime of the vacuum operation (in microseconds).                               |
| row_delta        | bigint    | Difference in the total number of table rows before and after the vacuum.                |

| Column name          | Data type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sortedrow_delta      | bigint    | Difference in the number of sorted table rows before and after the vacuum.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| block_delta          | integer   | Difference in block count for the table before and after the vacuum.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| max_merge_partitions | integer   | This column is used for performance analysis and represents the maximum number of partitions that vacuum can process for the table per merge phase iteration. (Vacuum sorts the unsorted region into one or more sorted partitions. Depending on the number of columns in the table and the current Amazon Redshift configuration, the merge phase can process a maximum number of partitions in a single merge iteration. The merge phase will still work if the number of sorted partitions exceeds the maximum number of merge partitions, but more merge iterations will be required.) |

## Sample query

The following query returns statistics for vacuum operations on three different tables. The SALES table was vacuumed twice.

```
select table_name, xid, sort_partitions as parts, merge_increments as merges,
elapsed_time, row_delta, sortedrow_delta as sorted_delta, block_delta
from svv_vacuum_summary
order by xid;

table_ | xid | parts|merges| elapsed_ | row_ | sorted_ | block_
name | | | | time | delta | delta | delta
-----+-----+-----+-----+-----+-----+-----+-----+
users | 2985 | 1 | 1 | 61919653 | 0 | 49990 | 20
category| 3982 | 1 | 1 | 24136484 | 0 | 11 | 0
sales | 3992 | 2 | 1 | 71736163 | 0 | 1207192 | 32
sales | 4000 | 1 | 1 | 15363010 | -851648 | -851648 | -140
(4 rows)
```

## SYS monitoring views

*Monitoring views* are system views in Amazon Redshift that are used to monitor query and workload resource usage of provisioned clusters and serverless workgroups. These views are located in the pg\_catalog schema. To display the information provided by these views, run SQL SELECT statements.

`SYS_SERVERLESS_USAGE` gathers usage data for Amazon Redshift Serverless only.

### Topics

- [SYS\\_CONNECTION\\_LOG \(p. 1466\)](#)
- [SYS\\_COPY\\_JOB \(preview\) \(p. 1468\)](#)
- [SYS\\_DATASHARE\\_CHANGE\\_LOG \(p. 1469\)](#)
- [SYS\\_DATASHARE\\_USAGE\\_CONSUMER \(p. 1471\)](#)
- [SYS\\_DATASHARE\\_USAGE\\_PRODUCER \(p. 1471\)](#)

- [SYS\\_EXTERNAL\\_QUERY\\_DETAIL \(p. 1472\)](#)
- [SYS\\_LOAD\\_DETAIL \(p. 1474\)](#)
- [SYS\\_LOAD\\_ERROR\\_DETAIL \(p. 1476\)](#)
- [SYS\\_LOAD\\_HISTORY \(p. 1477\)](#)
- [SYS\\_QUERY\\_DETAIL \(p. 1480\)](#)
- [SYS\\_QUERY\\_HISTORY \(p. 1483\)](#)
- [SYS\\_SERVERLESS\\_USAGE \(p. 1488\)](#)
- [SYS\\_STREAM\\_SCAN\\_ERRORS \(p. 1490\)](#)
- [SYS\\_STREAM\\_SCAN\\_STATES \(p. 1490\)](#)
- [SYS\\_UNLOAD\\_DETAIL \(p. 1492\)](#)
- [SYS\\_UNLOAD\\_HISTORY \(p. 1493\)](#)

## SYS\_CONNECTION\_LOG

Logs authentication attempts and connections and disconnections.

This view is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name     | Data type      | Description                                           |
|-----------------|----------------|-------------------------------------------------------|
| event           | character(50)  | Connection or authentication event.                   |
| record_time     | timestamp      | Time the event occurred.                              |
| remote_host     | character(32)  | Name or IP address of remote host.                    |
| remote_port     | character(32)  | Port number for remote host.                          |
| session_id      | integer        | Process ID associated with the statement.             |
| database_name   | character(50)  | Database name.                                        |
| user_name       | character(50)  | Username.                                             |
| auth_method     | character(32)  | Authentication method.                                |
| duration        | integer        | Duration of connection in microseconds.               |
| ssl_version     | character(50)  | Secure Sockets Layer (SSL) version.                   |
| ssl_cipher      | character(128) | SSL cipher.                                           |
| mtu             | integer        | Maximum transmission unit (MTU).                      |
| ssl_compression | character(64)  | SSL compression type.                                 |
| ssl_expansion   | character(64)  | SSL expansion type.                                   |
| iam_auth_guid   | character(36)  | The IAM authentication ID for the CloudTrail request. |

| Column name       | Data type      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| application_name  | character(250) | The initial or updated name of the application for a session.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| driver_version    | character(64)  | The version of ODBC or JDBC driver that connects to your Amazon Redshift cluster from your third-party SQL client tools.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| os_version        | character(64)  | The version of the operating system that is on the client machine that connects to your Amazon Redshift cluster.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| plugin_name       | character(32)  | The name of the plugin used to connect to your Amazon Redshift cluster.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| protocol_version  | integer        | The internal protocol version that the Amazon Redshift driver uses when establishing its connection with the server. The protocol versions are negotiated between the driver and server. The version describes the features available. Valid values include: <ul style="list-style-type: none"> <li>• 0 (BASE_SERVER_PROTOCOL_VERSION)</li> <li>• 1 (EXTENDED_RESULT_METADATA_SERVER_PROTOCOL_VERSION)               <ul style="list-style-type: none"> <li>– To save a round trip per query, the server sends extra result set metadata information.</li> </ul> </li> <li>• 2 (BINARY_PROTOCOL_VERSION) – Depending on the data type of the result set, the server sends data in binary format.</li> <li>• 3 (EXTENDED2_RESULT_METADATA_SERVER_PROTOCOL_VERSION)               <ul style="list-style-type: none"> <li>– The server sends case sensitivity (collation) information of a column.</li> </ul> </li> </ul> |
| global_session_id | character(36)  | The globally unique identifier for the current session. The session ID persists through node failure restarts.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

## Sample queries

To view the details for open connections, run the following query.

```
select record_time, user_name, database_name, remote_host, remote_port
from sys_connection_log
where event = 'initiating session'
and session_id not in
(select session_id from sys_connection_log
where event = 'disconnecting session')
order by 1 desc;

record_time | user_name | database_name | remote_host | remote_port
-----+-----+-----+-----+
2014-11-06 20:30:06 | rdsdb | dev | [local] |
2014-11-06 20:29:37 | test001 | test | 10.49.42.138 | 11111
2014-11-05 20:30:29 | rdsdb | dev | 10.49.42.138 | 33333
2014-11-05 20:28:35 | rdsdb | dev | [local] |
(4 rows)
```

The following example reflects a failed authentication attempt and a successful connection and disconnection.

```
select event, record_time, remote_host, user_name
from sys_connection_log order by record_time;

 event | record_time | remote_host | user_name
-----+-----+-----+-----+
authentication failure | 2012-10-25 14:41:56.96391 | 10.49.42.138 | john
authenticated | 2012-10-25 14:42:10.87613 | 10.49.42.138 | john
initiating session | 2012-10-25 14:42:10.87638 | 10.49.42.138 | john
disconnecting session | 2012-10-25 14:42:19.95992 | 10.49.42.138 | john
(4 rows)
```

The following example shows the version of the ODBC driver, the operating system on the client machine, and the plugin used to connect to the Amazon Redshift cluster. In this example, the plugin used is for standard ODBC driver authentication using a login name and password.

```
select driver_version, os_version, plugin_name from sys_connection_log;

driver_version | os_version | plugin_name
-----+-----+-----+
+-----+
Amazon Redshift ODBC Driver 1.4.15.0001 | Darwin 18.7.0 x86_64 | none
Amazon Redshift ODBC Driver 1.4.15.0001 | Linux 4.15.0-101-generic x86_64 | none
```

The following example shows the version of the operating system on the client machine, the driver version, and the protocol version.

```
select os_version, driver_version, protocol_version from sys_connection_log;

os_version | driver_version | protocol_version
-----+-----+-----+
+-----+
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
```

## SYS\_COPY\_JOB (preview)

*This is prerelease documentation for auto-copy (SQL COPY JOB), which is in preview release. The documentation and the feature are both subject to change. We recommend that you use this feature only in test environments, and not in production environments. Public preview will end on May 1, 2023. Preview clusters and preview serverless workgroups and namespaces will be removed automatically two weeks after the end of the preview. For preview terms and conditions, see [Betas and Previews in AWS Service Terms](#).*

Use SYS\_COPY\_JOB to view details of COPY JOB commands.

This view contains the COPY JOB commands that have been created.

SYS\_COPY\_JOB is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

## Table columns

| Column name      | Data type      | Description                                                                                                    |
|------------------|----------------|----------------------------------------------------------------------------------------------------------------|
| job_id           | bigint         | The copy job identifier.                                                                                       |
| job_name         | character(128) | The name of the copy job.                                                                                      |
| iam_role         | character(128) | The IAM role specified in the COPY statement.                                                                  |
| job_text         | character(256) | The parameters of the COPY statement.                                                                          |
| is_auto          | integer        | Indicates whether the COPY JOB is automatically run by Amazon Redshift. A 1 indicates true, 0 indicates false. |
| on_error_suspend | integer        | This information is for internal use only.                                                                     |

## SYS\_DATASHARE\_CHANGE\_LOG

Records the consolidated view for tracking changes to datashares on both producer and consumer clusters.

Superusers can see all rows; regular users can see only their own data.

## Table columns

| Column name       | Data type    | Description                                              |
|-------------------|--------------|----------------------------------------------------------|
| user_id           | integer      | The ID of the user taking the action.                    |
| user_name         | varchar(128) | The name of the user taking the action.                  |
| session_id        | integer      | The ID of the process.                                   |
| transaction_id    | bigint       | The ID of the transaction.                               |
| share_id          | integer      | The ID of the datashare affected.                        |
| share_name        | varchar(128) | The name of the datashare.                               |
| source_database   | integer      | The ID of the database to which the datashare belongs.   |
| source_database   | varchar(128) | The name of the database to which the datashare belongs. |
| consumer_database | integer      | The ID of the database imported from the datashare.      |
| consumer_database | varchar(128) | The name of the database imported from the datashare.    |
| arn               | varchar(192) | The ARN of the resource backing the imported database.   |
| record_time       | timestamp    | The timestamp of the action.                             |

| Column name        | Data type    | Description                                                                                                                                                                                                                                                                                                 |
|--------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| action             | varchar(128) | The action being run. Possible values are CREATE DATASHARE, DROP DATASHARE, GRANT ALTER, REVOKE ALTER, GRANT SHARE, REVOKE SHARE, ALTER ADD, ALTER REMOVE, ALTER SET, GRANT USAGE, REVOKE USAGE, CREATE DATABASE, GRANT, or REVOKE USAGE on a shared database, DROP SHARED DATABASE, ALTER SHARED DATABASE. |
| status             | integer      | The status of the action. Possible values are SUCCESS and ERROR-ERROR CODE.                                                                                                                                                                                                                                 |
| share_object_type  | varchar(64)  | The type of database object that was added to or removed from the datashare. Possible values are schemas, tables, columns, functions, and views. This is a field for the producer cluster.                                                                                                                  |
| share_object_id    | integer      | The ID of database object that was added to or removed from the datashare. This is a field for the producer cluster.                                                                                                                                                                                        |
| share_object_name  | varchar(128) | The name of database object that was added to or removed from the datashare. This is a field for the producer cluster.                                                                                                                                                                                      |
| target_user_type   | varchar(16)  | The type of users or groups that a privilege was granted to. This is a field for both the producer and consumer cluster.                                                                                                                                                                                    |
| target_user_id     | integer      | The ID of users or groups that a privilege was granted to. This is a field for both the producer and consumer cluster.                                                                                                                                                                                      |
| target_user_name   | varchar(128) | The name of users or groups that a privilege was granted to. This is a field for both the producer and consumer cluster.                                                                                                                                                                                    |
| consumer_account   | varchar(16)  | The account ID of the data consumer. This is a field for the producer cluster.                                                                                                                                                                                                                              |
| consumer_namespace | varchar(64)  | The namespace of the data consumer account. This is a field for the producer cluster.                                                                                                                                                                                                                       |
| producer_account   | varchar(16)  | The account ID of the producer account that the datashare belongs to. This is a field for the consumer cluster.                                                                                                                                                                                             |
| producer_namespace | varchar(64)  | The namespace of the product account that the datashare belongs to. This is a field for the consumer cluster.                                                                                                                                                                                               |
| attribute_name     | varchar(64)  | The name of an attribute of the datashare or shared database.                                                                                                                                                                                                                                               |
| attribute_value    | varchar(128) | The value of an attribute of the datashare or shared database.                                                                                                                                                                                                                                              |
| message            | varchar(512) | The error message when an action fails.                                                                                                                                                                                                                                                                     |

## Sample queries

The following example shows a SYS\_DATASHARE\_CHANGE\_LOG view.

```
SELECT DISTINCT action
FROM sys_datashare_change_log
WHERE share_object_name LIKE 'ticket%';

 action

 "ALTER DATASHARE ADD"
```

## SYS\_DATASHARE\_USAGE\_CONSUMER

Records the activity and usage of datashares. This view is only relevant on the consumer cluster.

SYS\_DATASHARE\_USAGE\_CONSUMER is visible to all users.

### Table columns

| Column name     | Data type    | Description                                           |
|-----------------|--------------|-------------------------------------------------------|
| user_id         | integer      | The ID of the user issuing the request.               |
| session_id      | integer      | The ID of the leader process running the query.       |
| transaction_id  | bigint       | The context of the current transaction.               |
| request_id      | varchar(50)  | The unique ID of the requested API call.              |
| request_type    | varchar(25)  | The type of the request made to the producer cluster. |
| transaction_uid | varchar(50)  | The unique ID of the transaction.                     |
| record_time     | timestamp    | The time when the action is recorded.                 |
| status          | integer      | The status of the requested API call.                 |
| error           | varchar(512) | The message for an error.                             |

### Sample queries

The following example shows a SYS\_DATASHARE\_USAGE\_CONSUMER view.

```
SELECT request_type, status, trim(error) AS error
FROM sys_datashare_usage_consumer
 request_type | status | error
-----+-----+-----+
 "GET RELATION" | 0 |
```

## SYS\_DATASHARE\_USAGE\_PRODUCER

Records the activity and usage of datashares. This view is only relevant on the producer cluster.

SYS\_DATASHARE\_USAGE\_PRODUCER is visible to all users.

### Table columns

| Column name | Data type    | Description                              |
|-------------|--------------|------------------------------------------|
| share_id    | integer      | The object ID (OID) of the datashare.    |
| share_name  | varchar(128) | The name of the datashare.               |
| request_id  | varchar(50)  | The unique ID of the requested API call. |

| Column name             | Data type    | Description                                                                                                                 |
|-------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------|
| request_type            | varchar(25)  | The type of the request made to the producer cluster.                                                                       |
| object_type             | varchar(64)  | The type of the object being shared from the datashare. Possible values are schemas, tables, columns, functions, and views. |
| object_oid              | integer      | The ID of the object being shared from the datashare.                                                                       |
| object_name             | varchar(128) | The name of the object being shared from the datashare.                                                                     |
| consumer_account        | varchar(16)  | The account of the consumer account that the datashare is shared to.                                                        |
| consumer_namespace      | varchar(64)  | The namespace of the consumer account that the datashare is shared to.                                                      |
| consumer_transaction_id | varchar(50)  | The unique transaction ID of the statement on the consumer cluster.                                                         |
| record_time             | timestamp    | The time when the action is recorded.                                                                                       |
| status                  | integer      | The status of the datashare.                                                                                                |
| error                   | varchar(512) | The message for an error.                                                                                                   |

## Sample queries

The following example shows a SYS\_DATASHARE\_USAGE\_PRODUCER view.

```
SELECT DISTINCT request_type
FROM sys_datashare_usage_producer
WHERE object_name LIKE 'tickit%';

request_type

"GET RELATION"
```

## SYS\_EXTERNAL\_QUERY\_DETAIL

Use SYS\_EXTERNAL\_QUERY\_DETAIL to view details for queries at a segment level. Each row represents a segment from a particular WLM query with details like the number of rows processed, number of bytes processed, and partition info of external tables in Amazon S3. Each row in this view will also have a corresponding entry in the SYS\_QUERY\_DETAIL view, except this view has more detail information related to external query processing.

SYS\_EXTERNAL\_QUERY\_DETAIL is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

### Table columns

| Column name | Data type | Description                                         |
|-------------|-----------|-----------------------------------------------------|
| user_id     | integer   | The identifier of the user who submitted the query. |
| query_id    | bigint    | The query identifier of the external query.         |

| Column name          | Data type       | Description                                                                                                |
|----------------------|-----------------|------------------------------------------------------------------------------------------------------------|
| transaction_id       | bigint          | The transaction identifier.                                                                                |
| child_query_sequence | integer         | The sequence of the rewritten user query. Starts with 0, similar to segment_id.                            |
| segment_id           | integer         | The segment identifier of the query segment.                                                               |
| source_type          | character(32)   | The data source type of the query, it could be S3 for Redshift Spectrum, PG for federated query.           |
| start_time           | timestamp       | The time when the query began.                                                                             |
| end_time             | timestamp       | The time when the query completed.                                                                         |
| duration             | bigint          | The amount of time (microseconds) spent on the query.                                                      |
| total_partitions     | integer         | The number of partitions an Amazon S3 query required.                                                      |
| qualified_partitions | integer         | The number of partitions an Amazon S3 query scanned.                                                       |
| scanned_files        | bigint          | The number of Amazon S3 files scanned.                                                                     |
| returned_rows        | bigint          | The number of scanned rows for an Amazon S3 query, or the number of returned rows for a federated query.   |
| returned_bytes       | bigint          | The number of scanned bytes for an Amazon S3 query, or the number of returned bytes for a federated query. |
| file_format          | text            | The file format of Amazon S3 files.                                                                        |
| file_location        | text            | The Amazon S3 location of external table.                                                                  |
| external_query_text  | text            | The segment level query text for a federated query.                                                        |
| warning_message      | character(4000) | The warning message displayed when the query runs.                                                         |

## Sample queries

The following query shows the external query details.

```

SELECT query_id,
 segment_id,
 start_time,
 end_time,
 total_partitions,
 qualified_partitions,
 scanned_files,
 returned_rows,
 returned_bytes,
 trim(external_query_text) query_text,
 trim(file_location) file_location
 FROM sys_external_query_detail
 ORDER BY query_id, start_time DESC
 LIMIT 2;

```

Sample output.

| query_id | segment_id | start_time                 | end_time                   | total_partitions | qualified_partitions | scanned_files | returned_rows | returned_bytes | query_text | file_location |
|----------|------------|----------------------------|----------------------------|------------------|----------------------|---------------|---------------|----------------|------------|---------------|
| 763251   | 0          | 2022-02-15 22:32:23.312448 | 2022-02-15 22:32:24.036023 | 3                | 3                    | 3             | 38203         | 2683414        |            |               |
| 763254   | 0          | 2022-02-15 22:32:40.17103  | 2022-02-15 22:32:40.839313 | 3                | 3                    | 3             | 38203         | 2683414        |            |               |

## SYS\_LOAD\_DETAIL

Returns information to track or troubleshoot a data load.

This view records the progress of each data file as it is loaded into a database table.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name   | Data type      | Description                                                                                                                                                                                                                               |
|---------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| user_id       | integer        | ID of the user who generated the entry.                                                                                                                                                                                                   |
| query_id      | integer        | Query ID.                                                                                                                                                                                                                                 |
| file_name     | character(256) | File name to be loaded.                                                                                                                                                                                                                   |
| bytes_scanned | integer        | The number of bytes scanned from the file in Amazon S3.                                                                                                                                                                                   |
| lines_scanned | integer        | Number of lines scanned from the load file. This number may not match the number of rows that are actually loaded. For example, the load may scan but tolerate a number of bad records, based on the MAXERROR option in the COPY command. |
| record_time   | timestamp      | Time that this entry was last updated.                                                                                                                                                                                                    |

## Sample queries

The following example returns details for the last COPY operation.

```
select query_id, trim(file_name) as file, record_time
from sys_load_detail
where query_id = pg_last_copy_id();

query_id | file | record_time
-----+-----+-----+
28554 | s3://dw-tickit/category_pipe.txt | 2013-11-01 17:14:52.648486
(1 row)
```

The following query contains entries for a fresh load of the tables in the TICKIT database:

```
select query_id, trim(file_name), record_time
from sys_load_detail
where file_name like '%tickit%' order by query_id;

query_id | btrim | record_time
-----+-----+-----+
22475 | tickit/allusers_pipe.txt | 2013-02-08 20:58:23.274186
22478 | tickit/venue_pipe.txt | 2013-02-08 20:58:25.070604
22480 | tickit/category_pipe.txt| 2013-02-08 20:58:27.333472
22482 | tickit/date2008_pipe.txt| 2013-02-08 20:58:28.608305
22485 | tickit/allevents_pipe.txt| 2013-02-08 20:58:29.99489
22487 | tickit/listings_pipe.txt| 2013-02-08 20:58:37.632939
22593 | tickit/allusers_pipe.txt| 2013-02-08 21:04:08.400491
22596 | tickit/venue_pipe.txt | 2013-02-08 21:04:10.056055
22598 | tickit/category_pipe.txt| 2013-02-08 21:04:11.465049
22600 | tickit/date2008_pipe.txt| 2013-02-08 21:04:12.461502
22603 | tickit/allevents_pipe.txt| 2013-02-08 21:04:14.785124
22605 | tickit/listings_pipe.txt| 2013-02-08 21:04:20.170594
(12 rows)
```

The fact that a record is written to the log file for this system view does not mean that the load committed successfully as part of its containing transaction. To verify load commits, query the STL\_UNUTILITYTEXT view and look for the COMMIT record that corresponds with a COPY transaction. For example, this query joins SYS\_LOAD\_DETAIL and STL\_QUERY based on a subquery against STL\_UNUTILITYTEXT:

```
select l.query_id,rtrim(l.file_name),q.xid
from sys_load_detail l, stl_query q
where l.query_id=q.query
and exists
(select xid from stl_unutilitytext where xid=q.xid and rtrim("text")='COMMIT');

query_id | rtrim | xid
-----+-----+-----+
22600 | tickit/date2008_pipe.txt | 68311
22480 | tickit/category_pipe.txt| 68066
7508 | allusers_pipe.txt | 23365
7552 | category_pipe.txt | 23415
7576 | allevents_pipe.txt | 23429
7516 | venue_pipe.txt | 23390
7604 | listings_pipe.txt | 23445
22596 | tickit/venue_pipe.txt | 68309
22605 | tickit/listings_pipe.txt| 68316
22593 | tickit/allusers_pipe.txt| 68305
22485 | tickit/allevents_pipe.txt| 68071
7561 | allevents_pipe.txt | 23429
```

|           |                           |       |
|-----------|---------------------------|-------|
| 7541      | category_pipe.txt         | 23415 |
| 7558      | date2008_pipe.txt         | 23428 |
| 22478     | tickit/venue_pipe.txt     | 68065 |
| 526       | date2008_pipe.txt         | 2572  |
| 7466      | allusers_pipe.txt         | 23365 |
| 22482     | tickit/date2008_pipe.txt  | 68067 |
| 22598     | tickit/category_pipe.txt  | 68310 |
| 22603     | tickit/allevents_pipe.txt | 68315 |
| 22475     | tickit/allusers_pipe.txt  | 68061 |
| 547       | date2008_pipe.txt         | 2572  |
| 22487     | tickit/listings_pipe.txt  | 68072 |
| 7531      | venue_pipe.txt            | 23390 |
| 7583      | listings_pipe.txt         | 23445 |
| (25 rows) |                           |       |

## SYS\_LOAD\_ERROR\_DETAIL

Use SYS\_LOAD\_ERROR\_DETAIL to view details of COPY command errors. Each row represents a COPY command. It contains both running and finished COPY commands.

SYS\_LOAD\_ERROR\_DETAIL is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

### Table columns

| Column name    | Data type      | Description                                                                                                                                     |
|----------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| user_id        | integer        | The identifier of the user who submitted the copy.                                                                                              |
| query_id       | bigint         | The query identifier of the copy.                                                                                                               |
| transaction_id | bigint         | The transaction identifier.                                                                                                                     |
| session_id     | integer        | The process identifier of the process running the copy.                                                                                         |
| database_name  | character(64)  | The name of the database the user was connected to when the copy was issued.                                                                    |
| table_id       | integer        | The table identifier.                                                                                                                           |
| start_time     | timestamp      | The time (UTC) when the copy began.                                                                                                             |
| file_name      | character(256) | The complete path to the input file to load.                                                                                                    |
| line_number    | bigint         | The line number in the load file with the error. When you load a JSON file, the line number of the last line of the JSON object with the error. |
| column_name    | character(127) | The field with the error.                                                                                                                       |
| column_type    | character(10)  | The data type of the field with the error.                                                                                                      |

| Column name   | Data type      | Description                                                                                                                                                                                        |
|---------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| column_length | character(10)  | The column length, if applicable. This field is populated when the data type has a limit length. For example, for a column with a data type of "character(3)", this column contains the value "3." |
| position      | integer        | The position of the error in the field.                                                                                                                                                            |
| error_code    | integer        | The error code.                                                                                                                                                                                    |
| error_message | character(512) | The explanation of the error.                                                                                                                                                                      |

## Sample queries

The following query shows the load error details of copy command for specific query.

```
SELECT query_id,
 table_id,
 start_time,
 trim(file_name) AS file_name,
 trim(column_name) AS column_name,
 trim(column_type) AS column_type,
 trim(error_message) AS error_message
 FROM sys_load_error_detail
 WHERE query_id = 762949
 ORDER BY start_time
LIMIT 10;
```

Sample output.

| query_id         | table_id | start_time                 | file_name                                      |
|------------------|----------|----------------------------|------------------------------------------------|
|                  |          |                            | error_message                                  |
| 762949           | 137885   | 2022-02-15 22:14:46.759151 | s3://load-test/copyfail/                       |
| wrong_format_000 | id       | int4                       | Invalid digit, Value 'a', Pos 0, Type: Integer |
| 762949           | 137885   | 2022-02-15 22:14:46.759151 | s3://load-test/copyfail/                       |
| wrong_format_001 | id       | int4                       | Invalid digit, Value 'a', Pos 0, Type: Integer |

## SYS\_LOAD\_HISTORY

Use SYS\_LOAD\_HISTORY to view details of COPY commands. Each row represents a COPY command with accumulated statistics for some of the fields. It contains both running and finished COPY commands.

SYS\_LOAD\_HISTORY is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

## Table columns

| Column name        | Data type | Description                                                                                                                                                        |
|--------------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| user_id            | integer   | The identifier of the user who submitted the copy.                                                                                                                 |
| query_id           | bigint    | The query identifier of the copy.                                                                                                                                  |
| transaction_id     | bigint    | The transaction identifier.                                                                                                                                        |
| session_id         | integer   | The process identifier of the process running the copy.                                                                                                            |
| database_name      | text      | The name of the database the user was connected to when the operation was issued.                                                                                  |
| status             | text      | The status of the copy. Valid values are <code>running</code> , <code>completed</code> , <code>aborted</code> .                                                    |
| table_name         | text      | The name of the table copying into.                                                                                                                                |
| start_time         | timestamp | The time when the copy began.                                                                                                                                      |
| end_time           | timestamp | The time when the copy completed.                                                                                                                                  |
| duration           | bigint    | The amount of time (microseconds) spent in the COPY command.                                                                                                       |
| data_source        | text      | The Amazon S3 location of files input to copy.                                                                                                                     |
| file_format        | text      | The source file format. Formats include <code>csv</code> , <code>txt</code> , <code>json</code> , <code>avro</code> , <code>orc</code> , or <code>parquet</code> . |
| loaded_rows        | bigint    | The number of rows copied to a table.                                                                                                                              |
| loaded_bytes       | bigint    | The number of bytes copied to a table.                                                                                                                             |
| source_file_count  | integer   | The number of files count in source files.                                                                                                                         |
| source_file_bytes  | bigint    | The number of bytes in source files.                                                                                                                               |
| file_count_scanned | integer   | The number of scanned files from Amazon S3.                                                                                                                        |
| file_bytes_scanned | bigint    | The number of bytes scanned from the file in Amazon S3.                                                                                                            |
| error_count        | bigint    | The number of errors count.                                                                                                                                        |

| Column name | Data type | Description                                               |
|-------------|-----------|-----------------------------------------------------------|
| copy_job_id | bigint    | The copy job identifier. A 0 indicates no job identifier. |

## Sample queries

The following query shows the loaded rows, bytes, tables, and datasource of specific copy commands.

```
SELECT query_id,
 table_name,
 data_source,
 loaded_rows,
 loaded_bytes
 FROM sys_load_history
 WHERE query_id IN (6389,490791,441663,74374,72297)
 ORDER BY query_id,
 data_source DESC;
```

Sample output.

| query_id | table_name                 | data_source                                          |
|----------|----------------------------|------------------------------------------------------|
|          | loaded_rows   loaded_bytes |                                                      |
| 6389     | store_returns              | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| 72297    | store_returns/             | 287999764   1196240296158                            |
| 72297    | web_site                   | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| 72297    | web_site/                  | 54   43808                                           |
| 74374    | ship_mode                  | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| 74374    | ship_mode/                 | 20   1320                                            |
| 441663   | income_band                | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| 441663   | income_band/               | 20   2152                                            |
| 490791   | customer_address           | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| 490791   | customer_address/          | 6000000   722924305                                  |

The following query shows the loaded rows, bytes, tables, and datasource of copy commands.

```
SELECT query_id,
 table_name,
 data_source,
 loaded_rows,
 loaded_bytes
 FROM sys_load_history
 ORDER BY query_id DESC
 LIMIT 10;
```

Sample output.

| query_id | table_name                 | data_source                                          |
|----------|----------------------------|------------------------------------------------------|
|          | loaded_rows   loaded_bytes |                                                      |
| 491058   | web_site                   | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| 491058   | web_site/                  | 54   43808                                           |
| 490947   | web_sales                  | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| 490947   | web_sales/                 | 720000376   22971988122819                           |

|                        |                                                      |
|------------------------|------------------------------------------------------|
| 490923   web_returns   | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| web_returns/           | 71997522   96597496325                               |
| 490918   web_page      | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| web_page/              | 3000   1320                                          |
| 490907   warehouse     | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| warehouse/             | 20   1320                                            |
| 490902   time_dim      | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| time_dim/              | 86400   1320                                         |
| 490876   store_sales   | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| store_sales/           | 2879987999   151666241887933                         |
| 490870   store_returns | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| store_returns/         | 287999764   1196405607941                            |
| 490865   store         | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/ |
| store/                 | 1002   365507                                        |

The following query shows the daily loaded rows and bytes of the copy command.

```
SELECT date_trunc('day',start_time) AS exec_day,
 SUM.loaded_rows AS loaded_rows,
 SUM.loaded_bytes AS loaded_bytes
 FROM sys_load_history
 GROUP BY exec_day
 ORDER BY exec_day DESC;
```

Sample output.

| exec_day            | loaded_rows | loaded_bytes     |
|---------------------|-------------|------------------|
| 2022-01-20 00:00:00 | 6347386005  | 258329473070606  |
| 2022-01-19 00:00:00 | 19042158015 | 775198502204572  |
| 2022-01-18 00:00:00 | 38084316030 | 1550294469446883 |
| 2022-01-17 00:00:00 | 25389544020 | 1033271084791724 |
| 2022-01-16 00:00:00 | 19042158015 | 775222736252792  |
| 2022-01-15 00:00:00 | 19834245387 | 798122849155598  |
| 2022-01-14 00:00:00 | 75376544688 | 3077040926571384 |

## SYS\_QUERY\_DETAIL

Use SYS\_QUERY\_DETAIL to view details for queries at a step level. Each row represents a step from a particular WLM query with details. This view contains many types of queries such as DDL, DML, and utility commands (for example, copy and unload). Some columns might not be relevant depending on the query type. For example, external\_scanned\_bytes is not relevant to internal tables.

### Note

A query that does not have a corresponding rewritten query has an entry only in the SYS\_QUERY\_HISTORY view, not in the SYS\_QUERY\_DETAIL view.

SYS\_QUERY\_DETAIL is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

## Table columns

| Column name | Data type | Description                                         |
|-------------|-----------|-----------------------------------------------------|
| user_id     | integer   | The identifier of the user who submitted the query. |
| query_id    | bigint    | The query identifier.                               |

| Column name          | Data type      | Description                                                                                                     |
|----------------------|----------------|-----------------------------------------------------------------------------------------------------------------|
| child_query_sequence | integer        | The sequence of the rewritten user query, starting with 1.                                                      |
| stream_id            | integer        | The stream identifier of the query stream.                                                                      |
| segment_id           | integer        | The segment identifier of the query running segment.                                                            |
| step_id              | integer        | The step identifier in a segment.                                                                               |
| step_name            | text           | The step name in a segment. For example, scan, hash, agg, and others.                                           |
| table_id             | integer        | The table identifier for permanent table scans.                                                                 |
| table_name           | character(136) | The table name of the step that is being operated.                                                              |
| is_rrsscan           | character      | A value that indicates whether a step is a scan step. True (t) indicates that a range-restricted scan was used. |
| start_time           | timestamp      | The time when the query step began.                                                                             |
| end_time             | timestamp      | The time when the query step completed.                                                                         |
| duration             | bigint         | The amount of time (microseconds) spent on the step.                                                            |
| alert                | text           | The description of the alert event.                                                                             |
| input_bytes          | bigint         | The input bytes for the current step.                                                                           |
| input_rows           | bigint         | The input rows for the current step.                                                                            |
| output_bytes         | bigint         | The output bytes for the current step.                                                                          |
| output_rows          | bigint         | The output rows for the current step.                                                                           |
| blocks_read          | bigint         | The number of block the step read.                                                                              |
| blocks_write         | bigint         | The number of block the step wrote.                                                                             |

| Column name    | Data type | Description                                                                                                                                                             |
|----------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| local_read_IO  | bigint    | The number of blocks read from local disk cache.                                                                                                                        |
| remote_read_IO | bigint    | The number of blocks read from remote.                                                                                                                                  |
| data_skewness  | integer   | The skewness of output rows distribution among all steps. It is a number in the range of 0% to 100%. The larger the number, the more unbalanced is the distribution.    |
| time_skewness  | integer   | The skewness of execution time distribution among all steps. It is a number in the range of 0% to 100%. The larger the number, the more unbalanced is the distribution. |
| is_active      | character | The state of the query at the step level. Possible values are 't' that shows the step is actively running or 'f' that indicates the step completes running.             |

## Sample queries

The following example returns the output of SYS\_QUERY\_DETAIL.

The following query shows the query metadata detail at step level, including step name, input\_bytes, output\_bytes, input\_rows, output\_rows.

```
SELECT query_id,
 child_query_sequence,
 stream_id,
 segment_id,
 step_id,
 trim(step_name) AS step_name,
 duration,
 input_bytes,
 output_bytes,
 input_rows,
 output_rows
 FROM sys_query_detail
 WHERE query_id IN (193929)
 ORDER BY query_id,
 stream_id,
 segment_id,
 step_id DESC;
```

Sample output.

| query_id                             | child_query_sequence                 | stream_id                            | segment_id                           | step_id                              | step_name                            | duration                             | input_bytes                          | output_bytes                         | input_rows                           | output_rows                          |
|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|
| -----+-----+-----+-----+-----+-----+ | -----+-----+-----+-----+-----+-----+ | -----+-----+-----+-----+-----+-----+ | -----+-----+-----+-----+-----+-----+ | -----+-----+-----+-----+-----+-----+ | -----+-----+-----+-----+-----+-----+ | -----+-----+-----+-----+-----+-----+ | -----+-----+-----+-----+-----+-----+ | -----+-----+-----+-----+-----+-----+ | -----+-----+-----+-----+-----+-----+ | -----+-----+-----+-----+-----+-----+ |

|        |        |         |        |        |               |  |
|--------|--------|---------|--------|--------|---------------|--|
| 193929 |        | 2       | 0      | 0      | 3   hash      |  |
| 37144  | 0      | 9350272 | 0      | 292196 |               |  |
| 193929 |        | 5       | 0      | 0      | 3   hash      |  |
| 9492   | 0      | 23360   | 0      | 1460   |               |  |
| 193929 |        | 1       | 0      | 0      | 3   hash      |  |
| 46809  | 0      | 9350272 | 0      | 292196 |               |  |
| 193929 |        | 4       | 0      | 0      | 2   return    |  |
| 7685   | 0      | 896     | 0      | 112    |               |  |
| 193929 |        | 1       | 0      | 0      | 2   project   |  |
| 46809  | 0      | 0       | 0      | 292196 |               |  |
| 193929 |        | 2       | 0      | 0      | 2   project   |  |
| 37144  | 0      | 0       | 0      | 292196 |               |  |
| 193929 |        | 5       | 0      | 0      | 2   project   |  |
| 9492   | 0      | 0       | 0      | 1460   |               |  |
| 193929 |        | 3       | 0      | 0      | 2   return    |  |
| 11033  | 0      | 14336   | 0      | 112    |               |  |
| 193929 |        | 2       | 0      | 0      | 1   project   |  |
| 37144  | 0      | 0       | 0      | 292196 |               |  |
| 193929 |        | 1       | 0      | 0      | 1   project   |  |
| 46809  | 0      | 0       | 0      | 292196 |               |  |
| 193929 |        | 5       | 0      | 0      | 1   project   |  |
| 9492   | 0      | 0       | 0      | 1460   |               |  |
| 193929 |        | 3       | 0      | 0      | 1   aggregate |  |
| 11033  | 0      | 201488  | 0      | 14     |               |  |
| 193929 |        | 4       | 0      | 0      | 1   aggregate |  |
| 7685   | 0      | 28784   | 0      | 14     |               |  |
| 193929 |        | 5       | 0      | 0      | 0   scan      |  |
| 9492   | 0      | 23360   | 292196 | 1460   |               |  |
| 193929 |        | 4       | 0      | 0      | 0   scan      |  |
| 7685   | 0      | 1344    | 112    | 112    |               |  |
| 193929 |        | 2       | 0      | 0      | 0   scan      |  |
| 37144  | 0      | 7304900 | 292196 | 292196 |               |  |
| 193929 |        | 3       | 0      | 0      | 0   scan      |  |
| 11033  | 0      | 13440   | 112    | 112    |               |  |
| 193929 |        | 1       | 0      | 0      | 0   scan      |  |
| 46809  | 0      | 7304900 | 292196 | 292196 |               |  |
| 193929 |        | 5       | 0      | 0      | -1            |  |
| 9492   | 12288  | 0       | 0      | 0      |               |  |
| 193929 |        | 1       | 0      | 0      | -1            |  |
| 46809  | 16384  | 0       | 0      | 0      |               |  |
| 193929 |        | 2       | 0      | 0      | -1            |  |
| 37144  | 16384  | 0       | 0      | 0      |               |  |
| 193929 |        | 4       | 0      | 0      | -1            |  |
| 7685   | 28672  | 0       | 0      | 0      |               |  |
| 193929 |        | 3       | 0      | 0      | -1            |  |
| 11033  | 114688 | 0       | 0      | 0      |               |  |

## SYS\_QUERY\_HISTORY

Use SYS\_QUERY\_HISTORY to view details of user queries. Each row represents a user query with accumulated statistics for some of the fields. This view contains many types of queries, such as data definition language (DDL), data manipulation language (DML), copy, unload, and Amazon Redshift Spectrum. It contains both running and finished queries.

SYS\_QUERY\_HISTORY is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

## Table columns

| Column name      | Data type       | Description                                                                                                 |
|------------------|-----------------|-------------------------------------------------------------------------------------------------------------|
| user_id          | integer         | The identifier of the user who submitted the query.                                                         |
| query_id         | bigint          | The query identifier.                                                                                       |
| query_label      | character(320)  | The short name for the query.                                                                               |
| transaction_id   | bigint          | The transaction identifier.                                                                                 |
| session_id       | integer         | The process identifier of the process running the query.                                                    |
| database_name    | character(128)  | The name of the database the user was connected to when the query was issued.                               |
| query_type       | character(32)   | The type of query, such as, SELECT, INSERT, UPDATE, UNLOAD COPY, COMMAND, DDL, UTILITY, CTAS, and OTHER.    |
| status           | character(10)   | The status of the query. Valid values: planning, queued, running, returning, failed, canceled, and success. |
| result_cache_hit | Boolean         | Indicates whether the query matches the result cache.                                                       |
| start_time       | timestamp       | The time when the query began.                                                                              |
| end_time         | timestamp       | The time when the query completed.                                                                          |
| elapsed_time     | bigint          | The total amount of time (microseconds) spent on the query.                                                 |
| queue_time       | bigint          | The total time (microseconds) spent on the service class query queue.                                       |
| execution_time   | bigint          | The total time (microseconds) running in the service class.                                                 |
| error_message    | character(512)  | The reason a query failed.                                                                                  |
| returned_rows    | bigint          | The number of rows returned to the client.                                                                  |
| returned_bytes   | bigint          | The number of bytes returned to the client.                                                                 |
| query_text       | character(4000) | The query string. This string might be truncated.                                                           |

| Column name      | Data type      | Description                                                                                                                                                                                                                           |
|------------------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| redshift_version | character(256) | The Amazon Redshift version when the query ran.                                                                                                                                                                                       |
| usage_limit      | character(150) | List of usage limit IDs reached by the query.                                                                                                                                                                                         |
| compute_type     | varchar(32)    | Indicates whether the query runs on the main cluster or concurrency scaling cluster. Possible values are <code>primary</code> (query runs on the main cluster) or <code>primary-scale</code> (query runs on the concurrency cluster). |
| compile_time     | bigint         | The total time (microseconds) spent on compilation of the query.                                                                                                                                                                      |
| planning_time    | bigint         | The total time (microseconds) spent on planning of the query.                                                                                                                                                                         |
| lock_wait_time   | bigint         | The total time (microseconds) spent on waiting for relation lock.                                                                                                                                                                     |

## Sample queries

The following query returns running and queued queries.

```
SELECT user_id,
 query_id,
 transaction_id,
 session_id,
 status,
 trim(database_name) AS database_name,
 start_time,
 end_time,
 result_cache_hit,
 elapsed_time,
 queue_time,
 execution_time
 FROM sys_query_history
 WHERE status IN ('running','queued')
 ORDER BY start_time;
```

Sample output.

| user_id | query_id | transaction_id | session_id | status  | database_name | start_time                | end_time                   | result_cache_hit | elapsed_time | queue_time | execution_time |
|---------|----------|----------------|------------|---------|---------------|---------------------------|----------------------------|------------------|--------------|------------|----------------|
| 101     | 760705   | 852337         | 1073832321 | running | tpcds_1t      | 2022-02-15 19:03:19.67849 | 2022-02-15 19:03:19.739811 | f                | 61321        | 0          | 0              |

The following query returns the query start time, end time, queue time, elapsed time, planning time, and other metadata for a specific query.

```
SELECT user_id,
 query_id,
 transaction_id,
 session_id,
 status,
 trim(database_name) AS database_name,
 start_time,
 end_time,
 result_cache_hit,
 elapsed_time,
 queue_time,
 execution_time,
 planning_time,
 trim(query_text) as query_text
 FROM sys_query_history
 WHERE query_id = 3093;
```

Sample output.

| user_id | query_id | transaction_id | session_id | status     | database_name | start_time | end_time | result_cache_hit | elapsed_time | queue_time | execution_time | planning_time | query_text                                                                                                              |
|---------|----------|----------------|------------|------------|---------------|------------|----------|------------------|--------------|------------|----------------|---------------|-------------------------------------------------------------------------------------------------------------------------|
| 106     | 3093     |                | 11759      | 1073750146 | success       | dev        |          |                  |              |            |                |               | 2023-03-16 16:53:17.840214   2023-03-16 16:53:18.106588   f   266374   0   105725   136589   select count(*) from item; |

The following query lists the 10 most recent SELECT queries.

```
SELECT query_id,
 transaction_id,
 session_id,
 start_time,
 elapsed_time,
 queue_time,
 execution_time,
 returned_rows,
 returned_bytes
 FROM sys_query_history
 WHERE query_type = 'SELECT'
 ORDER BY start_time DESC limit 10;
```

Sample output.

| query_id | transaction_id | session_id | start_time                 | elapsed_time | queue_time | execution_time | returned_rows | returned_bytes |
|----------|----------------|------------|----------------------------|--------------|------------|----------------|---------------|----------------|
| 526532   | 61093          | 1073840313 | 2022-02-09 04:43:24.149603 | 520571       | 0          | 481293         | 1             | 3794           |
| 526520   | 60850          | 1073840313 | 2022-02-09 04:38:27.24875  | 635957       | 0          | 596601         | 1             | 3679           |
| 526508   | 60803          | 1073840313 | 2022-02-09 04:37:51.118835 | 563882       | 0          | 503135         | 5             | 17216          |

|                                                                                    |
|------------------------------------------------------------------------------------|
| 526505   0   589823   60763   1073840313   2022-02-09 04:36:48.636224   649337     |
| 526478   0   14544058   60730   1073840313   2022-02-09 04:36:11.741471   14611321 |
| 526467   0   16633767   60636   1073840313   2022-02-09 04:34:11.91463   16711367  |
| 511617   0   9899271   617946   1074009948   2022-01-20 06:21:54.44481   9937090   |
| 511603   0   7582500   617941   1074259415   2022-01-20 06:21:45.71744   8065081   |
| 511595   0   1014879   617935   1074128320   2022-01-20 06:21:44.030876   1051270  |
| 511584   0   485887   617931   1074030019   2022-01-20 06:21:42.764088   609033    |
|                                                                                    |

The following query shows the daily select query count and average query elapsed time.

```
SELECT date_trunc('day',start_time) AS exec_day,
 status,
 COUNT(*) AS query_cnt,
 AVG(datediff (microsecond,start_time,end_time)) AS elapsed_avg
 FROM sys_query_history
 WHERE query_type = 'SELECT'
 AND start_time >= '2022-01-14'
 AND start_time <= '2022-01-18'
 GROUP BY exec_day,
 status
 ORDER BY exec_day,
 status;
```

Sample output.

| exec_day            | status  | query_cnt | elapsed_avg |
|---------------------|---------|-----------|-------------|
| 2022-01-14 00:00:00 | success | 5253      | 56608048    |
| 2022-01-15 00:00:00 | success | 7004      | 56995017    |
| 2022-01-16 00:00:00 | success | 5253      | 57016363    |
| 2022-01-17 00:00:00 | success | 5309      | 55236784    |
| 2022-01-18 00:00:00 | success | 8092      | 54355124    |

The following query shows the daily query elapsed time performance.

```
SELECT distinct date_trunc('day',start_time) AS exec_day,
 query_count.cnt AS query_count,
 Percentile_cont(0.5) within group(ORDER BY elapsed_time) OVER (PARTITION BY
exec_day) AS P50_runtime,
 Percentile_cont(0.8) within group(ORDER BY elapsed_time) OVER (PARTITION BY
exec_day) AS P80_runtime,
 Percentile_cont(0.9) within group(ORDER BY elapsed_time) OVER (PARTITION BY
exec_day) AS P90_runtime,
 Percentile_cont(0.99) within group(ORDER BY elapsed_time) OVER (PARTITION BY
exec_day) AS P99_runtime,
 Percentile_cont(1.0) within group(ORDER BY elapsed_time) OVER (PARTITION BY
exec_day) AS max_runtime
 FROM sys_query_history
 LEFT JOIN (SELECT date_trunc('day',start_time) AS day, count(*) cnt
 FROM sys_query_history
 WHERE query_type = 'SELECT'
 GROUP by 1) query_count
 ON date_trunc('day',start_time) = query_count.day
 WHERE query_type = 'SELECT'
 ORDER BY exec_day;
```

Sample output.

| exec_day<br>  max_runtime          | query_count | p50_runtime | p80_runtime | p90_runtime | p99_runtime  |
|------------------------------------|-------------|-------------|-------------|-------------|--------------|
| 2022-01-14 00:00:00   1582078873.0 | 5253        | 16816922.0  | 69525096.0  | 158524917.8 | 486322477.52 |
| 2022-01-15 00:00:00   1696344792.0 | 7004        | 15896130.5  | 71058707.0  | 164314568.9 | 500331542.07 |
| 2022-01-16 00:00:00   1594793766.0 | 5253        | 15750451.0  | 72037082.2  | 159513733.4 | 480372059.24 |
| 2022-01-17 00:00:00   1521758640.0 | 5309        | 15394513.0  | 68881393.2  | 160254700.0 | 493372245.84 |
| 2022-01-18 00:00:00   1542783444.0 | 8092        | 15575286.5  | 68485955.4  | 154559572.5 | 463552685.39 |
| 2022-01-19 00:00:00   1693483241.0 | 5860        | 16648747.0  | 72470482.6  | 166485138.2 | 492038228.67 |
| 2022-01-20 00:00:00   1439319739.0 | 1751        | 15422072.0  | 69686381.0  | 162315385.0 | 497066615.00 |
| 2022-02-09 00:00:00   23168439.0   | 13          | 6382812.0   | 17616161.6  | 21197988.4  | 23021343.84  |

The following query shows the query type distribution.

```
SELECT query_type,
 COUNT(*) AS query_count
 FROM sys_query_history
 GROUP BY query_type
 ORDER BY query_count DESC;
```

Sample output.

| query_type | query_count |
|------------|-------------|
| UTILITY    | 134486      |
| SELECT     | 38537       |
| DDL        | 4832        |
| OTHER      | 768         |
| LOAD       | 768         |
| CTAS       | 748         |
| COMMAND    | 92          |

## SYS\_SERVERLESS\_USAGE

Use SYS\_SERVERLESS\_USAGE to view details of Amazon Redshift Serverless usage of resources. This system view doesn't apply to provisioned Amazon Redshift clusters.

This view contains the serverless usage summary including how much compute capacity is used to process queries and the amount of Amazon Redshift managed storage used at a 1-minute granularity. The compute capacity is measured in Redshift processing units (RPUs) and metered for the workloads that you run in RPU-seconds on a per-second basis. RPUs are used to process queries on the data loaded in the data warehouse, queried from an Amazon S3 data lake, or accessed from operational databases using a federated query. Amazon Redshift Serverless retains the information in SYS\_SERVERLESS\_USAGE for 7 days.

For examples on compute cost billing, see [Billing for Amazon Redshift Serverless](#).

SYS\_SERVERLESS\_USAGE is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

## Table columns

| Column name                   | Data type        | Description                                                                                                                                                                                                                                                                                                                                             |
|-------------------------------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| start_time                    | timestamp        | The time when the interval began.                                                                                                                                                                                                                                                                                                                       |
| end_time                      | timestamp        | The time when the interval completed.                                                                                                                                                                                                                                                                                                                   |
| compute_seconds               | double precision | The accumulated compute unit (RPU) seconds consumed during this time interval. This value accounts for the base RPU capacity allocated for the account.                                                                                                                                                                                                 |
| compute_capacity              | double precision | The average number of compute units (Redshift processing units, or RPUs) allocated during this time interval.<br><br>The compute_capacity value can be dynamically changed.                                                                                                                                                                             |
| data_storage                  | integer          | The average data storage space in MB used during this time interval.<br><br>Used data storage can change dynamically as data is loaded or deleted from the database.                                                                                                                                                                                    |
| cross_region_transferred_data | integer          | The accumulated data transferred for cross-Region data sharing in bytes during this time interval.                                                                                                                                                                                                                                                      |
| charged_seconds               | integer          | The accumulated compute unit (RPU) seconds charged during this time interval. This is computed after transactions end, and hence can be 0 while a transaction runs. Use charged_seconds to calculate cost for an Amazon Redshift Serverless workgroup. This value accounts for the RPU capacity allocated for the Amazon Redshift Serverless workgroup. |

### Example

To get the total charges for RPU hours used for a time interval by querying charged\_seconds, run the following query:

```
select trunc(start_time) "Day",
```

```
(sum(charged_seconds)/3600::double precision) * <Price for 1 RPU> as cost_incurred
from sys_serverless_usage
group by 1
order by 1
```

Note that there can be idle time during the interval. Idle time doesn't add to RPUs consumed.

## SYS\_STREAM\_SCAN\_ERRORS

Records errors for records loaded via streaming ingestion.

SYS\_STREAM\_SCAN\_ERRORS is visible only to the superuser.

### Table columns

| Column name      | Data type                   | Description                                                                                                                         |
|------------------|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| external_schema  | character(128)              | The name of the Kinesis stream or Amazon MSK topic's schema. It is case sensitive.                                                  |
| stream_name      | character(255)              | The name of the stream or topic. It is case sensitive.                                                                              |
| mv_name          | character(128)              | The name of the associated materialized view. Empty if none. It is case sensitive.                                                  |
| transaction_id   | bigint                      | The transaction ID.                                                                                                                 |
| query_id         | bigint                      | The query ID.                                                                                                                       |
| stream_timestamp | timestamp                   | The type of the stream timestamp. It is case sensitive.                                                                             |
| stream_timestamp | timestamp without time zone | The time when the record arrived.                                                                                                   |
| record_time      | timestamp without time zone | The time when the error message was logged.                                                                                         |
| partition_id     | character(128)              | The partition/shard id. It is case sensitive.                                                                                       |
| position         | character(128)              | The position of the record. This corresponds with the sequence number in Kinesis or the offset in Amazon MSK. It is case sensitive. |
| error_code       | integer                     | The error code.                                                                                                                     |
| error_reason     | character(128)              | The error reason. It is case sensitive.                                                                                             |

## SYS\_STREAM\_SCAN\_STATES

Records scan states for records loaded via streaming ingestion.

SYS\_STREAM\_SCAN\_STATES is visible only to the superuser.

## Table columns

| Column name        | Data type                   | Description                                                                                                                                                |
|--------------------|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| external_schema    | character(128)              | The external schema name. It is case sensitive.                                                                                                            |
| stream_name        | character(256)              | The stream name. It is case sensitive.                                                                                                                     |
| mv_name            | character(128)              | The name of the associated materialized view. Empty if none. It is case sensitive.                                                                         |
| transaction_id     | bigint                      | The transaction ID.                                                                                                                                        |
| query_id           | bigint                      | The query ID.                                                                                                                                              |
| record_time        | timestamp without time zone | The time when the data was logged.                                                                                                                         |
| partition_id       | character(128)              | The partition or shard id. It is case sensitive.                                                                                                           |
| latest_position    | character(128)              | The position of the last record read in the batch. This corresponds with the sequence number in Kinesis or the offset in Amazon MSK. It is case sensitive. |
| scanned_rows       | bigint                      | The number of records that were scanned in the batch.                                                                                                      |
| skipped_rows       | bigint                      | The number of records that were skipped in the batch.                                                                                                      |
| scanned_bytes      | bigint                      | The number of bytes that were scanned in the batch.                                                                                                        |
| stream_record_time | timestamp without time zone | Kinesis or Amazon MSK arrival time for the earliest record in the batch.                                                                                   |
| stream_record_time | timestamp without time zone | Kinesis or Amazon MSK arrival time for the latest record in the batch.                                                                                     |

The following query shows stream and topic data for specific queries.

```
select
 query_id,mv_name::varchar,external_schema_name::varchar,stream_name::varchar,sum(scanned_rows)
 total_records,
 sum(scanned_bytes) total_bytes from sys_stream_scan_states where query in (5401180,8601939)
 group by 1,2,3,4;

 query_id | mv_name | external_schema_name | stream_name | total_records | total_bytes
-----+-----+-----+-----+-----+-----+
+-----+
 5401180 | kinesistest | kinesis | kinesisstream | 1493255696 |
 3209006490704
 8601939 | msktest | msk | mskstream | 14677023 |
 31056580668
(2 rows)
```

## SYS\_UNLOAD\_DETAIL

Use SYS\_UNLOAD\_DETAIL to view details of an UNLOAD operation. It records one row for each file created by an UNLOAD statement. For example, if an UNLOAD creates 12 files, SYS\_UNLOAD\_DETAIL will contain 12 corresponding rows.

SYS\_UNLOAD\_DETAIL is visible to all users. Superusers can see all rows; regular users can see only their own data.

### Table columns

| Column name    | Data type        | Description                                                    |
|----------------|------------------|----------------------------------------------------------------|
| user_id        | integer          | The identifier of the user who generated the entry.            |
| query_id       | integer          | The query identifier of the UNLOAD command.                    |
| session_id     | integer          | The ID of the process associated with the query statement.     |
| transaction_id | bigint           | The ID of the transaction associated with the query statement. |
| file_name      | character (1280) | The complete Amazon S3 object path for the file.               |
| start_time     | timestamp        | The time when the transaction began.                           |
| end_time       | timestamp        | The time when the transaction completed.                       |
| line_count     | bigint           | The number of lines (rows) unloaded to the file.               |
| transfer_size  | bigint           | The number of bytes transferred.                               |
| file_format    | character (10)   | The file format of the unloaded files.                         |

### Sample queries

The following query shows the unloaded query details, including format, rows, and file count of unload command.

```
select query_id, substring(file_name, 0, 50), transfer_size, file_format from
sys_unload_detail;
```

Sample output.

| query_id  <br>file_format | substring | transfer_size |
|---------------------------|-----------|---------------|
|                           |           |               |

| 9272     | s3://my-bucket/my_unload_doc_venue0000_part_00.gz | 395886 | Text |  |
|----------|---------------------------------------------------|--------|------|--|
| 9272     | s3://my-bucket/my_unload_doc_venue0001_part_00.gz | 406444 | Text |  |
| 9272     | s3://my-bucket/my_unload_doc_venue0002_part_00.gz | 409431 | Text |  |
| 9272     | s3://my-bucket/my_unload_doc_venue0003_part_00.gz | 403051 | Text |  |
| 9272     | s3://my-bucket/my_unload_doc_venue0004_part_00.gz | 413592 | Text |  |
| 9272     | s3://my-bucket/my_unload_doc_venue0005_part_00.gz | 395689 | Text |  |
| (6 rows) |                                                   |        |      |  |

## SYS\_UNLOAD\_HISTORY

Use SYS\_UNLOAD\_HISTORY to view details of UNLOAD commands. Each row represents a UNLOAD command with accumulated statistics for some of the fields. It contains both running and finished UNLOAD commands.

SYS\_UNLOAD\_HISTORY is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access.

### Table columns

| Column name       | Data type | Description                                                                       |
|-------------------|-----------|-----------------------------------------------------------------------------------|
| user_id           | integer   | The identifier of the user who submitted the unload.                              |
| query_id          | bigint    | The query identifier of the UNLOAD command.                                       |
| transaction_id    | bigint    | The transaction identifier.                                                       |
| session_id        | integer   | The process identifier of the process running the unload.                         |
| database_name     | text      | The name of the database the user was connected to when the operation was issued. |
| status            | text      | The status of the UNLOAD command. Valid values are running and completed.         |
| start_time        | timestamp | The time when the unload began.                                                   |
| end_time          | timestamp | The time when the unload completed.                                               |
| duration          | bigint    | The amount of time (microseconds) spent in the UNLOAD command.                    |
| file_format       | text      | The file format of the output files.                                              |
| compression_type  | text      | The compression type.                                                             |
| unloaded_location | text      | The Amazon S3 location of unloaded files.                                         |

| Column name          | Data type | Description                              |
|----------------------|-----------|------------------------------------------|
| unloaded_rows        | bigint    | The number of rows.                      |
| unloaded_files_count | bigint    | The file count of the output file.       |
| unloaded_files_size  | bigint    | The file size of the output file.        |
| error_message        | text      | The error message of the UNLOAD command. |

## Sample queries

The following query shows the unloaded query details, including format, rows, and file count of unload command.

```
SELECT query_id,
 file_format,
 start_time,
 duration,
 unloaded_rows,
 unloaded_files_count
FROM sys_unload_history
ORDER BY query_id,
 file_format limit 100;
```

Sample output.

| query_id | file_format | start_time                 | duration | unloaded_rows | unloaded_files_count |
|----------|-------------|----------------------------|----------|---------------|----------------------|
| 527067   | Text        | 2022-02-09 05:18:35.844452 | 5932478  | 10            | 1                    |

## System monitoring (provisioned only)

The following system tables and views can be queried on provisioned clusters. STL and STV tables and views contain a subset of data found in several of the system tables. These provide quicker and easier access to commonly queried data found in those tables.

SVCS views provide details about queries on both the main and concurrency scaling clusters. SVL views provide information only for queries run on the main cluster.

### Topics

- [STL views for logging \(p. 1494\)](#)
- [STV tables for snapshot data \(p. 1577\)](#)
- [SVCS views for main and concurrency scaling clusters \(p. 1610\)](#)
- [SVL views for main cluster \(p. 1628\)](#)

## STL views for logging

STL system views are generated from Amazon Redshift log files to provide a history of the system.

These files reside on every node in the data warehouse cluster. The STL views take the information from the logs and format them into usable views for system administrators.

**Log retention** – STL system views retain seven days of log history. Log retention is guaranteed for all cluster sizes and node types, and it isn't affected by changes in cluster workload. Log retention also isn't affected by cluster status, such as when the cluster is paused. You have less than seven days of log history only in the case where the cluster is new. Retaining logs doesn't require any customer action, but if you want to store log data for more than 7 days, you have to periodically copy it to other tables or unload it to Amazon S3.

## Topics

- [STL\\_AGGR \(p. 1496\)](#)
- [STL\\_ALERT\\_EVENT\\_LOG \(p. 1498\)](#)
- [STL\\_ANALYZE \(p. 1499\)](#)
- [STL\\_ANALYZE\\_COMPRESSION \(p. 1501\)](#)
- [STL\\_BCAST \(p. 1502\)](#)
- [STL\\_COMMIT\\_STATS \(p. 1503\)](#)
- [STL\\_CONNECTION\\_LOG \(p. 1505\)](#)
- [STL\\_DDLTEXT \(p. 1507\)](#)
- [STL\\_DELETE \(p. 1509\)](#)
- [STL\\_DISK\\_FULL\\_DIAG \(p. 1511\)](#)
- [STL\\_DIST \(p. 1512\)](#)
- [STL\\_ERROR \(p. 1513\)](#)
- [STL\\_EXPLAIN \(p. 1514\)](#)
- [STL\\_FILE\\_SCAN \(p. 1516\)](#)
- [STL\\_HASH \(p. 1517\)](#)
- [STL\\_HASHJOIN \(p. 1519\)](#)
- [STL\\_INSERT \(p. 1520\)](#)
- [STL\\_LIMIT \(p. 1521\)](#)
- [STL\\_LOAD\\_COMMITS \(p. 1523\)](#)
- [STL\\_LOAD\\_ERRORS \(p. 1526\)](#)
- [STL\\_LOADERROR\\_DETAIL \(p. 1528\)](#)
- [STL\\_MERGE \(p. 1530\)](#)
- [STL\\_MERGEJOIN \(p. 1531\)](#)
- [STL\\_MV\\_STATE \(p. 1532\)](#)
- [STL\\_NESTLOOP \(p. 1534\)](#)
- [STL\\_PARSE \(p. 1535\)](#)
- [STL\\_PLAN\\_INFO \(p. 1536\)](#)
- [STL\\_PROJECT \(p. 1538\)](#)
- [STL\\_QUERY \(p. 1539\)](#)
- [STL\\_QUERY\\_METRICS \(p. 1542\)](#)
- [STL\\_QUERYTEXT \(p. 1544\)](#)
- [STL\\_REPLACEMENTS \(p. 1547\)](#)
- [STL\\_RESTARTED\\_SESSIONS \(p. 1547\)](#)
- [STL\\_RETURN \(p. 1548\)](#)
- [STL\\_S3CLIENT \(p. 1549\)](#)
- [STL\\_S3CLIENT\\_ERROR \(p. 1551\)](#)

- [STL\\_SAVE \(p. 1553\)](#)
- [STL\\_SCAN \(p. 1554\)](#)
- [STL\\_SCHEMA\\_QUOTA\\_VIOLATIONS \(p. 1557\)](#)
- [STL\\_SESSIONS \(p. 1557\)](#)
- [STL\\_SORT \(p. 1559\)](#)
- [STL\\_SSHCLIENT\\_ERROR \(p. 1560\)](#)
- [STL\\_STREAM\\_SEGS \(p. 1560\)](#)
- [STL\\_TR\\_CONFLICT \(p. 1561\)](#)
- [STL\\_UNDONE \(p. 1562\)](#)
- [STL\\_UNIQUE \(p. 1563\)](#)
- [STL\\_UNLOAD\\_LOG \(p. 1564\)](#)
- [STL\\_USAGE\\_CONTROL \(p. 1565\)](#)
- [STL\\_USERLOG \(p. 1566\)](#)
- [STL.UtilityText \(p. 1567\)](#)
- [STL\\_VACUUM \(p. 1570\)](#)
- [STL\\_WINDOW \(p. 1573\)](#)
- [STL\\_WLM\\_ERROR \(p. 1574\)](#)
- [STL\\_WLM\\_RULE\\_ACTION \(p. 1574\)](#)
- [STL\\_WLM\\_QUERY \(p. 1575\)](#)

## [STL\\_AGGR](#)

Analyzes aggregate execution steps for queries. These steps occur during execution of aggregate functions and GROUP BY clauses.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                            |
|-------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                                |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                          |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                          |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                              |
| step        | integer   | Query step that ran.                                                                                                                                                                   |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds.<br>For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds.<br>For example: <b>2009-06-12 11:29:19.131358</b> . |

| Column name  | Data type    | Description                                                                                                                                                                                                                                                                                                      |
|--------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tasknum      | integer      | Number of the query task process that was assigned to run the step.                                                                                                                                                                                                                                              |
| rows         | bigint       | Total number of rows that were processed.                                                                                                                                                                                                                                                                        |
| bytes        | bigint       | Size, in bytes, of all the output rows for the step.                                                                                                                                                                                                                                                             |
| slots        | integer      | Number of hash buckets.                                                                                                                                                                                                                                                                                          |
| occupied     | integer      | Number of slots that contain records.                                                                                                                                                                                                                                                                            |
| maxlength    | integer      | Size of the largest slot.                                                                                                                                                                                                                                                                                        |
| tbl          | integer      | Table ID.                                                                                                                                                                                                                                                                                                        |
| is_diskbased | character(1) | If true (t), the query was run as a disk-based operation. If false (f), the query was run in memory.                                                                                                                                                                                                             |
| workmem      | bigint       | Number of bytes of working memory assigned to the step.                                                                                                                                                                                                                                                          |
| type         | character(6) | The type of step. Valid values are: <ul style="list-style-type: none"> <li>HASHED. Indicates that the step used grouped, unsorted aggregation.</li> <li>PLAIN. Indicates that the step used ungrouped, scalar aggregation.</li> <li>SORTED. Indicates that the step used grouped, sorted aggregation.</li> </ul> |
| resizes      | integer      | This information is for internal use only.                                                                                                                                                                                                                                                                       |
| flushable    | integer      | This information is for internal use only.                                                                                                                                                                                                                                                                       |

## Sample queries

Returns information about aggregate execution steps for SLICE 1 and TBL 239.

```
select query, segment, bytes, slots, occupied, maxlength, is_diskbased, workmem, type
from stl_aggr where slice=1 and tbl=239
order by rows
limit 10;
```

| query | segment | bytes | slots   | occupied | maxlength | is_diskbased | workmem   | type   |
|-------|---------|-------|---------|----------|-----------|--------------|-----------|--------|
| 562   | 1       | 0     | 4194304 | 0        | 0         | f            | 383385600 | HASHED |
| 616   | 1       | 0     | 4194304 | 0        | 0         | f            | 383385600 | HASHED |
| 546   | 1       | 0     | 4194304 | 0        | 0         | f            | 383385600 | HASHED |
| 547   | 0       | 8     | 0       | 0        | 0         | f            | 0         | PLAIN  |
| 685   | 1       | 32    | 4194304 | 1        | 0         | f            | 383385600 | HASHED |

|           |   |    |         |   |       |  |           |
|-----------|---|----|---------|---|-------|--|-----------|
| 652       | 0 | 8  | 0       | 0 | 0   f |  | 0         |
| PLAIN     |   |    |         |   |       |  |           |
| 680       | 0 | 8  | 0       | 0 | 0   f |  | 0         |
| PLAIN     |   |    |         |   |       |  |           |
| 658       | 0 | 8  | 0       | 0 | 0   f |  | 0         |
| PLAIN     |   |    |         |   |       |  |           |
| 686       | 0 | 8  | 0       | 0 | 0   f |  | 0         |
| PLAIN     |   |    |         |   |       |  |           |
| 695       | 1 | 32 | 4194304 | 1 | 0   f |  | 383385600 |
| HASHED    |   |    |         |   |       |  |           |
| (10 rows) |   |    |         |   |       |  |           |

## STL\_ALERT\_EVENT\_LOG

Records an alert when the query optimizer identifies conditions that might indicate performance issues. Use the STL\_ALERT\_EVENT\_LOG view to identify opportunities to improve query performance.

A query consists of multiple segments, and each segment consists of one or more steps. For more information, see [Query processing \(p. 463\)](#).

STL\_ALERT\_EVENT\_LOG is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type       | Description                                                                                                                                                                        |
|-------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer         | ID of the user who generated the entry.                                                                                                                                            |
| query       | integer         | Query ID. The query column can be used to join other system tables and views.                                                                                                      |
| slice       | integer         | Number that identifies the slice where the query was running.                                                                                                                      |
| segment     | integer         | Number that identifies the query segment.                                                                                                                                          |
| step        | integer         | Query step that ran.                                                                                                                                                               |
| pid         | integer         | Process ID associated with the statement and slice. The same query might have multiple PIDs if it runs on multiple slices.                                                         |
| xid         | bigint          | Transaction ID associated with the statement.                                                                                                                                      |
| event       | character(1024) | Description of the alert event.                                                                                                                                                    |
| solution    | character(1024) | Recommended solution.                                                                                                                                                              |
| event_time  | timestamp       | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |

### Usage notes

You can use the STL\_ALERT\_EVENT\_LOG to identify potential issues in your queries, then follow the practices in [Tuning query performance \(p. 463\)](#) to optimize your database design and rewrite your queries. STL\_ALERT\_EVENT\_LOG records the following alerts:

- **Missing statistics**

Statistics are missing. Run ANALYZE following data loads or significant updates and use STATUPDATE with COPY operations. For more information, see [Amazon Redshift best practices for designing queries \(p. 21\)](#).

- **Nested loop**

A nested loop is usually a Cartesian product. Evaluate your query to ensure that all participating tables are joined efficiently.

- **Very selective filter**

The ratio of rows returned to rows scanned is less than 0.05. Rows scanned is the value of `rows_pre_user_filter` and rows returned is the value of rows in the [STL\\_SCAN \(p. 1554\)](#) system view. Indicates that the query is scanning an unusually large number of rows to determine the result set. This can be caused by missing or incorrect sort keys. For more information, see [Working with sort keys \(p. 60\)](#).

- **Excessive ghost rows**

A scan skipped a relatively large number of rows that are marked as deleted but not vacuumed, or rows that have been inserted but not committed. For more information, see [Vacuuming tables \(p. 110\)](#).

- **Large distribution**

More than 1,000,000 rows were redistributed for hash join or aggregation. For more information, see [Working with data distribution styles \(p. 49\)](#).

- **Large broadcast**

More than 1,000,000 rows were broadcast for hash join. For more information, see [Working with data distribution styles \(p. 49\)](#).

- **Serial execution**

A DS\_DIST\_ALL\_INNER redistribution style was indicated in the query plan, which forces serial execution because the entire inner table was redistributed to a single node. For more information, see [Working with data distribution styles \(p. 49\)](#).

## Sample queries

The following query shows alert events for four queries.

```
SELECT query, substring(event,0,25) as event,
substring(solution,0,25) as solution,
trim(event_time) as event_time from stl_alert_event_log order by query;

query | event | solution | event_time
-----+-----+-----+
+-----+
 6567 | Missing query planner statist | Run the ANALYZE command | 2014-01-03 18:20:58
 7450 | Scanned a large number of del | Run the VACUUM command to rec| 2014-01-03 21:19:31
 8406 | Nested Loop Join in the query | Review the join predicates to| 2014-01-04 00:34:22
29512 | Very selective query filter:r | Review the choice of sort key| 2014-01-06 22:00:00
(4 rows)
```

## STL\_ANALYZE

Records details for [ANALYZE \(p. 663\)](#) operations.

This view is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name       | Data type      | Description                                                                                                                                                        |
|-------------------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid            | integer        | The ID of the user who generated the entry.                                                                                                                        |
| xid               | long           | The transaction ID.                                                                                                                                                |
| database          | char(30)       | The database name.                                                                                                                                                 |
| table_id          | integer        | The table ID.                                                                                                                                                      |
| status            | char(15)       | The result of the analyze command. Possible values are Full, Skipped, and PredicateColumn.                                                                         |
| rows              | double         | The total number of rows in the table.                                                                                                                             |
| modified_rows     | double         | The total number of rows that were modified since the last ANALYZE operation.                                                                                      |
| threshold_percent | integer        | The value of the analyze_threshold_percent parameter.                                                                                                              |
| is_auto           | char(1)        | The value is true (t) if the operation included an Amazon Redshift analyze operation by default. The value is false (f) if the ANALYZE command was run explicitly. |
| starttime         | timestamp      | The time in UTC that the analyze operation started running.                                                                                                        |
| endtime           | timestamp      | The time in UTC that the analyze operation finished running.                                                                                                       |
| prevtime          | timestamp      | The time in UTC that the table was previously analyzed.                                                                                                            |
| num_predicate     | integer        | The current number of predicate columns in the table.                                                                                                              |
| num_new_predicate | integer        | The number of new predicate columns in the table since the previous analyze operation.                                                                             |
| is_background     | character(1)   | The value is true (t) if the analysis was run by an automatic analyze operation. Otherwise, the value is false (f).                                                |
| auto_analyze_ph   | character(100) | Reserved for internal use.                                                                                                                                         |

## Sample queries

The following example joins STV\_TBL\_PERM to show the table name and execution details.

```
select distinct a.xid, trim(t.name) as name, a.status, a.rows, a.modified_rows,
a.starttime, a.endtime
from stl_analyze a
join stv_tbl_perm t on t.id=a.table_id
where name = 'users'
order by starttime;
```

| xid | name | status | rows | modified_rows | starttime | endtime |
|-----|------|--------|------|---------------|-----------|---------|
|-----|------|--------|------|---------------|-----------|---------|

| userid   | tbl   | mode    | start_time | xid | old_encoding | new_encoding | sample_size | analyze_start_time  | analyze_end_time | analyze_duration |
|----------|-------|---------|------------|-----|--------------|--------------|-------------|---------------------|------------------|------------------|
| 1582     | users | Full    | 22:02:28   |     | 49990        | 49990        | 1           | 2016-09-22 22:02:23 | 2016-09-22       |                  |
| 244287   | users | Full    | 22:51:01   |     | 24992        | 74988        | 1           | 2016-10-04 22:50:58 | 2016-10-04       |                  |
| 244712   | users | Full    | 22:56:07   |     | 49984        | 24992        | 1           | 2016-10-04 22:56:07 | 2016-10-04       |                  |
| 245071   | users | Skipped | 22:58:17   |     | 49984        | 0            | 1           | 2016-10-04 22:58:17 | 2016-10-04       |                  |
| 245439   | users | Skipped | 23:00:13   |     | 49984        | 1982         | 1           | 2016-10-04 23:00:13 | 2016-10-04       |                  |
| (5 rows) |       |         |            |     |              |              |             |                     |                  |                  |

## STL\_ANALYZE\_COMPRESSION

Records details for compression analysis operations during COPY or ANALYZE COMPRESSION commands.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name  | Data type      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid       | integer        | The ID of the user who generated the entry.                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| start_time   | timestamp      | The time when the compression analysis operation started.                                                                                                                                                                                                                                                                                                                                                                                                                                |
| xid          | bigint         | The transaction ID of the compression analysis operation.                                                                                                                                                                                                                                                                                                                                                                                                                                |
| tbl          | integer        | The table ID of the table that was analyzed.                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| tablename    | character(128) | The name of the table that was analyzed.                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| col          | integer        | The index of the column in the table that was analyzed to determine the compression encoding.                                                                                                                                                                                                                                                                                                                                                                                            |
| old_encoding | character(15)  | The encoding type before compression analysis.                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| new_encoding | character(15)  | The encoding type after compression analysis.                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| mode         | character(14)  | <p>The possible values are:</p> <p><b>PRESET</b></p> <p>Specifies that the new_encoding is determined by the Amazon Redshift COPY command based on the column data type. No data is sampled.</p> <p><b>ON</b></p> <p>Specifies that the new_encoding is determined by the Amazon Redshift COPY command based on an analysis of sample data.</p> <p><b>ANALYZE ONLY</b></p> <p>Specifies that the new_encoding is determined by the Amazon Redshift ANALYZE COMPRESSION command based</p> |

| Column name | Data type | Description                                                                                      |
|-------------|-----------|--------------------------------------------------------------------------------------------------|
|             |           | on an analysis of sample data. However, the encoding type of the analyzed column is not changed. |

## Sample queries

The following example inspects the details of compression analysis on the `lineitem` table by the last `COPY` command run in the same session.

```
select xid, tbl, btrim(tablename) as tablename, col, old_encoding, new_encoding, mode
from stl_analyze_compression
where xid = (select xid from stl_query where query = pg_last_copy_id()) order by col;

+-----+-----+-----+-----+-----+-----+-----+
| xid | tbl | tablename | col | old_encoding | new_encoding | mode |
+-----+-----+-----+-----+-----+-----+-----+
8196	248126	lineitem	0	mostly32	mostly32	ON
8196	248126	lineitem	1	mostly32	lzo	ON
8196	248126	lineitem	2	lzo	delta32k	ON
8196	248126	lineitem	3	delta	delta	ON
8196	248126	lineitem	4	bytedict	bytedict	ON
8196	248126	lineitem	5	mostly32	mostly32	ON
8196	248126	lineitem	6	delta	delta	ON
8196	248126	lineitem	7	delta	delta	ON
8196	248126	lineitem	8	lzo	zstd	ON
8196	248126	lineitem	9	runlength	zstd	ON
8196	248126	lineitem	10	delta	lzo	ON
8196	248126	lineitem	11	delta	delta	ON
8196	248126	lineitem	12	delta	delta	ON
8196	248126	lineitem	13	bytedict	zstd	ON
8196	248126	lineitem	14	bytedict	zstd	ON
8196	248126	lineitem	15	text255	zstd	ON
+-----+-----+-----+-----+-----+-----+-----+
(16 rows)
```

## STL\_BCAST

Logs information about network activity during execution of query steps that broadcast data. Network traffic is captured by numbers of rows, bytes, and packets that are sent over the network during a given step on a given slice. The duration of the step is the difference between the logged start and end times.

To identify broadcast steps in a query, look for `bcast` labels in the `SVL_QUERY_SUMMARY` view or run the `EXPLAIN` command and then look for step attributes that include `bcast`.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type | Description                                                                   |
|-------------|-----------|-------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                       |
| query       | integer   | Query ID. The query column can be used to join other system tables and views. |
| slice       | integer   | Number that identifies the slice where the query was running.                 |

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows        | bigint    | Total number of rows that were processed.                                                                                                                                           |
| bytes       | bigint    | Size, in bytes, of all the output rows for the step.                                                                                                                                |
| packets     | integer   | Total number of packets sent over the network.                                                                                                                                      |

### Sample queries

The following example returns broadcast information for the queries where there are one or more packets, and the difference between the start and end of the query was one second or more.

```
select query, slice, step, rows, bytes, packets, datediff(seconds, starttime, endtime)
from stl_bcast
where packets>0 and datediff(seconds, starttime, endtime)>0;
```

| query    | slice | step | rows | bytes | packets | date_diff |
|----------|-------|------|------|-------|---------|-----------|
| 453      | 2     | 5    | 1    | 264   | 1       | 1         |
| 798      | 2     | 5    | 1    | 264   | 1       | 1         |
| 1408     | 2     | 5    | 1    | 264   | 1       | 1         |
| 2993     | 0     | 5    | 1    | 264   | 1       | 1         |
| 5045     | 3     | 5    | 1    | 264   | 1       | 1         |
| 8073     | 3     | 5    | 1    | 264   | 1       | 1         |
| 8163     | 3     | 5    | 1    | 264   | 1       | 1         |
| 9212     | 1     | 5    | 1    | 264   | 1       | 1         |
| 9873     | 1     | 5    | 1    | 264   | 1       | 1         |
| (9 rows) |       |      |      |       |         |           |

## STL\_COMMIT\_STATS

Provides metrics related to commit performance, including the timing of the various stages of commit and the number of blocks committed. Query `STL_COMMIT_STATS` to determine what portion of a transaction was spent on commit and how much queuing is occurring.

This view is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name      | Data type | Description                                                                     |
|------------------|-----------|---------------------------------------------------------------------------------|
| xid              | bigint    | Transaction id being committed.                                                 |
| node             | integer   | Node number. -1 is the leader node.                                             |
| startqueue       | timestamp | Start of queueing for commit.                                                   |
| startwork        | timestamp | Start of commit.                                                                |
| endflush         | timestamp | End of dirty block flush phase.                                                 |
| endstage         | timestamp | End of metadata staging phase.                                                  |
| endlocal         | timestamp | End of local commit phase.                                                      |
| startglobal      | timestamp | Start of global phase.                                                          |
| endtime          | timestamp | End of the commit.                                                              |
| queuelen         | bigint    | Number of transactions that were ahead of this transaction in the commit queue. |
| permblocks       | bigint    | Number of existing permanent blocks at the time of this commit.                 |
| newblocks        | bigint    | Number of new permanent blocks at the time of this commit.                      |
| dirtyblocks      | bigint    | Number of blocks that had to be written as part of this commit.                 |
| headers          | bigint    | Number of block headers that had to be written as part of this commit.          |
| numxids          | integer   | The number of active DML transactions.                                          |
| oldestxid        | bigint    | The XID of the oldest active DML transaction.                                   |
| extwritelatency  | bigint    | This information is for internal use only.                                      |
| metadatawritten  | int       | This information is for internal use only.                                      |
| tombstonedblocks | bigint    | This information is for internal use only.                                      |
| tossedblocks     | bigint    | This information is for internal use only.                                      |
| batched_by       | bigint    | This information is for internal use only.                                      |

## Sample query

```
select node, datediff(ms,startqueue,startwork) as queue_time,
datediff(ms, startwork, endtime) as commit_time, queuelen
from stl_commit_stats
where xid = 2574
order by node;

node | queue_time | commit_time | queuelen
-----+-----+-----+-----+
-1 | 0 | 617 | 0
0 | 444950725641 | 616 | 0
```

|                  |     |   |
|------------------|-----|---|
| 1   444950725636 | 616 | 0 |
|------------------|-----|---|

## STL\_CONNECTION\_LOG

Logs authentication attempts and connections and disconnections.

This view is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name      | Data type      | Description                                                                                                                                                                                                                                                                                                                                      |
|------------------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| event            | character(50)  | Connection or authentication event.                                                                                                                                                                                                                                                                                                              |
| recordtime       | timestamp      | Time the event occurred.                                                                                                                                                                                                                                                                                                                         |
| remotehost       | character(32)  | Name or IP address of remote host.                                                                                                                                                                                                                                                                                                               |
| remoteport       | character(32)  | Port number for remote host.                                                                                                                                                                                                                                                                                                                     |
| pid              | integer        | Process ID associated with the statement.                                                                                                                                                                                                                                                                                                        |
| dbname           | character(50)  | Database name.                                                                                                                                                                                                                                                                                                                                   |
| username         | character(50)  | User name.                                                                                                                                                                                                                                                                                                                                       |
| authmethod       | character(32)  | Authentication method.                                                                                                                                                                                                                                                                                                                           |
| duration         | integer        | Duration of connection in microseconds.                                                                                                                                                                                                                                                                                                          |
| sslversion       | character(50)  | Secure Sockets Layer (SSL) version.                                                                                                                                                                                                                                                                                                              |
| sslcipher        | character(128) | SSL cipher.                                                                                                                                                                                                                                                                                                                                      |
| mtu              | integer        | Maximum transmission unit (MTU).                                                                                                                                                                                                                                                                                                                 |
| sslcompression   | character(64)  | SSL compression type.                                                                                                                                                                                                                                                                                                                            |
| sslexpansion     | character(64)  | SSL expansion type.                                                                                                                                                                                                                                                                                                                              |
| iamauthguid      | character(36)  | The IAM authentication ID for the CloudTrail request.                                                                                                                                                                                                                                                                                            |
| application_name | character(250) | The initial or updated name of the application for a session.                                                                                                                                                                                                                                                                                    |
| os_version       | character(64)  | The version of the operating system that is on the client machine that connects to your Amazon Redshift cluster.                                                                                                                                                                                                                                 |
| driver_version   | character(64)  | The version of ODBC or JDBC driver that connects to your Amazon Redshift cluster from your third-party SQL client tools.                                                                                                                                                                                                                         |
| plugin_name      | character(32)  | The name of the plugin used to connect to your Amazon Redshift cluster.                                                                                                                                                                                                                                                                          |
| protocol_version | integer        | The internal protocol version that the Amazon Redshift driver uses when establishing its connection with the server. The protocol versions is negotiated between the driver and server. The version describes the features available. Valid values include: <ul style="list-style-type: none"> <li>• 0 (BASE_SERVER_PROTOCOL_VERSION)</li> </ul> |

| Column name | Data type     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|             |               | <ul style="list-style-type: none"> <li>• 1<br/>(EXTENDED_RESULT_METADATA_SERVER_PROTOCOL_VERSION)<br/>– To save a round trip per query, the server sends extra result set metadata information.</li> <li>• 2 (BINARY_PROTOCOL_VERSION) – Depending on the data type of the result set, the server sends data in binary format.</li> <li>• 3<br/>(EXTENDED2_RESULT_METADATA_SERVER_PROTOCOL_VERSION)<br/>– The server sends case sensitivity (collation) information of a column.</li> </ul> |
| sessionid   | character(36) | The globally unique identifier for the current session. The session ID persists through node failure restarts.                                                                                                                                                                                                                                                                                                                                                                              |

## Sample queries

To view the details for open connections, run the following query.

```
select recordtime, username, dbname, remotehost, remoteport
from stl_connection_log
where event = 'initiating session'
and pid not in
(select pid from stl_connection_log
where event = 'disconnecting session')
order by 1 desc;

recordtime | username | dbname | remotehost | remoteport
-----+-----+-----+-----+
2014-11-06 20:30:06 | rdsdb | dev | [local] |
2014-11-06 20:29:37 | test001 | test | 10.49.42.138 | 11111
2014-11-05 20:30:29 | rdsdb | dev | 10.49.42.138 | 33333
2014-11-05 20:28:35 | rdsdb | dev | [local] |
(4 rows)
```

The following example reflects a failed authentication attempt and a successful connection and disconnection.

```
select event, recordtime, remotehost, username
from stl_connection_log order by recordtime;

event | recordtime | remotehost | username
-----+-----+-----+
authentication failure | 2012-10-25 14:41:56.96391 | 10.49.42.138 | john
authenticated | 2012-10-25 14:42:10.87613 | 10.49.42.138 | john
initiating session | 2012-10-25 14:42:10.87638 | 10.49.42.138 | john
disconnecting session | 2012-10-25 14:42:19.95992 | 10.49.42.138 | john
(4 rows)
```

The following example shows the version of the ODBC driver, the operating system on the client machine, and the plugin used to connect to the Amazon Redshift cluster. In this example, the plugin used is for standard ODBC driver authentication using a login name and password.

```
select driver_version, os_version, plugin_name from stl_connection_log;

driver_version | os_version | plugin_name
-----+-----+-----+
+-----+
Amazon Redshift ODBC Driver 1.4.15.0001 | Darwin 18.7.0 x86_64 | none
Amazon Redshift ODBC Driver 1.4.15.0001 | Linux 4.15.0-101-generic x86_64 | none
```

The following example shows the version of the operating system on the client machine, the driver version, and the protocol version.

```
select os_version, driver_version, protocol_version from stl_connection_log;

os_version | driver_version | protocol_version
-----+-----+-----+
+-----+
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
```

## STL\_DDLTEXT

Captures the following DDL statements that were run on the system.

These DDL statements include the following queries and objects:

- CREATE SCHEMA, TABLE, VIEW
- DROP SCHEMA, TABLE, VIEW
- ALTER SCHEMA, TABLE

See also [STL\\_QUERYTEXT \(p. 1544\)](#), [STL\\_UTILITYTEXT \(p. 1567\)](#), and [SVI\\_STATEMENTTEXT \(p. 1661\)](#). These views provide a timeline of the SQL commands that are run on the system; this history is useful for troubleshooting and for creating an audit trail of all system activities.

Use the STARTTIME and ENDTIME columns to find out which statements were logged during a given time period. Long blocks of SQL text are broken into lines 200 characters long; the SEQUENCE column identifies fragments of text that belong to a single statement.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type      | Description                                                                                                          |
|-------------|----------------|----------------------------------------------------------------------------------------------------------------------|
| userid      | integer        | ID of the user who generated the entry.                                                                              |
| xid         | bigint         | Transaction ID associated with the statement.                                                                        |
| pid         | integer        | Process ID associated with the statement.                                                                            |
| label       | character(320) | Either the name of the file used to run the query or a label defined with a SET QUERY_GROUP command. If the query is |

| Column name | Data type      | Description                                                                                                                                                                         |
|-------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|             |                | not file-based or the QUERY_GROUP parameter is not set, this field is blank.                                                                                                        |
| starttime   | timestamp      | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp      | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| sequence    | integer        | When a single statement contains more than 200 characters, additional rows are logged for that statement. Sequence 0 is the first row, 1 is the second, and so on.                  |
| text        | character(200) | SQL text, in 200-character increments. This field might contain special characters such as backslash (\) and newline (\n).                                                          |

## Sample queries

The following query returns records that include previously run DDL statements.

```
select xid, starttime, sequence, substring(text,1,40) as text
from stl_ddltext order by xid desc, sequence;
```

The following is sample output that shows four CREATE TABLE statements. The DDL statements appear in the text column, which is truncated for readability.

| xid      | starttime                  | sequence | text                                      |
|----------|----------------------------|----------|-------------------------------------------|
| 1806     | 2013-10-23 00:11:14.709851 | 0        | CREATE TABLE supplier ( s_suppkey int4 N  |
| 1806     | 2013-10-23 00:11:14.709851 | 1        | s_comment varchar(101) NOT NULL )         |
| 1805     | 2013-10-23 00:11:14.496153 | 0        | CREATE TABLE region ( r_regionkey int4 N  |
| 1804     | 2013-10-23 00:11:14.285986 | 0        | CREATE TABLE partsupp ( ps_partkey int8 N |
| 1803     | 2013-10-23 00:11:14.056901 | 0        | CREATE TABLE part ( p_partkey int8 NOT N  |
| 1803     | 2013-10-23 00:11:14.056901 | 1        | ner char(10) NOT NULL , p_retailprice nu  |
| (6 rows) |                            |          |                                           |

## Reconstructing Stored SQL

The following SQL lists rows stored in the text column of **STL\_DDLTEXT**. The rows are ordered by xid and sequence. If the original SQL was longer than 200 characters multiple rows, **STL\_DDLTEXT** can contain multiple rows by sequence.

```
SELECT xid, sequence, LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text)
END, '') WITHIN GROUP (ORDER BY sequence) as query_statement
FROM stl_ddltext GROUP BY xid, sequence ORDER BY xid, sequence;
```

| xid     | sequence | query_statement                                                                                                                                    |
|---------|----------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| 7886671 | 0        | create external schema schema_spectrum_udbh\nfrom data catalog \ndatabase 'spectrum_db_udbh'\niam_role ''\ncreate external database if not exists; |

```

7886752 0 CREATE EXTERNAL TABLE schema_spectrum_uddh.soccer_league\n(\n
league_rank smallint,\n prev_rank smallint,\n club_name varchar(15),\n league_name\n varchar(20),\n league_off decimal(6,2),\n le
7886752 1 ague_def decimal(6,2),\n league_spi decimal(6,2),\n league_nspli\n smallint\n)\nROW FORMAT DELIMITED \n FIELDS TERMINATED BY ',' \n LINES TERMINATED BY
'\\n\\l'\nstored as textfile\nLOCATION 's3://mybucket-spectrum-uddh/'\ntable properties
('skip.header.line.count='1');
...

```

To reconstruct the SQL stored in the text column of `STL_DDLTEXT`, run the following SQL statement. It puts together DDL statements from one or more segments in the text column. Before running the reconstructed SQL, replace any `(\n)` special characters with a new line in your SQL client. The results of the following `SELECT` statement puts together three rows in sequence order to reconstruct the SQL, in the `query_statement` field.

```

SELECT LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END) WITHIN GROUP
(ORDER BY sequence) as query_statement
FROM stl_ddltext GROUP BY xid, endtime order by xid, endtime;

```

```

query_statement

create external schema schema_spectrum_uddh\nfrom data catalog\ndatabase
'spectrum_db_uddh'\niam_role ''\ncreate external database if not exists;
CREATE EXTERNAL TABLE schema_spectrum_uddh.soccer_league\n(\n league_rank smallint,
\n prev_rank smallint,\n club_name varchar(15),\n league_name varchar(20),\n league_off decimal(6,2),\n league_def decimal(6,2),\n league_spi decimal(6,2),\n league_nspli smallint\n)\nROW FORMAT DELIMITED \n FIELDS TERMINATED BY ',' \n LINES
TERMINATED BY '\\n\\l'\nstored as textfile\nLOCATION 's3://mybucket-spectrum-uddh/'\ntable
properties ('skip.header.line.count='1');

```

## STL\_DELETE

Analyzes delete execution steps for queries.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                        |
|-------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                            |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                      |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                      |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                          |
| step        | integer   | Query step that ran.                                                                                                                                                               |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution, with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows        | bigint    | Total number of rows that were processed.                                                                                                                                           |
| tbl         | integer   | Table ID.                                                                                                                                                                           |

## Sample queries

In order to create a row in STL\_DELETE, the following example inserts a row into the EVENT table and then deletes it.

First, insert a row into the EVENT table and verify that it was inserted.

```
insert into event(eventid,venueid,catid,dateid,eventname)
values ((select max(eventid)+1 from event),95,9,1857,'Lollapalooza');
```

```
select * from event
where eventname='Lollapalooza'
order by eventid;
```

| eventid   | venueid | catid | dateid | eventname    | starttime           |
|-----------|---------|-------|--------|--------------|---------------------|
| 4274      | 102     | 9     | 1965   | Lollapalooza | 2008-05-01 19:00:00 |
| 4684      | 114     | 9     | 2105   | Lollapalooza | 2008-10-06 14:00:00 |
| 5673      | 128     | 9     | 1973   | Lollapalooza | 2008-05-01 15:00:00 |
| 5740      | 51      | 9     | 1933   | Lollapalooza | 2008-04-17 15:00:00 |
| 5856      | 119     | 9     | 1831   | Lollapalooza | 2008-01-05 14:00:00 |
| 6040      | 126     | 9     | 2145   | Lollapalooza | 2008-11-15 15:00:00 |
| 7972      | 92      | 9     | 2026   | Lollapalooza | 2008-07-19 19:30:00 |
| 8046      | 65      | 9     | 1840   | Lollapalooza | 2008-01-14 15:00:00 |
| 8518      | 48      | 9     | 1904   | Lollapalooza | 2008-03-19 15:00:00 |
| 8799      | 95      | 9     | 1857   | Lollapalooza |                     |
| (10 rows) |         |       |        |              |                     |

Now, delete the row that you added to the EVENT table and verify that it was deleted.

```
delete from event
where eventname='Lollapalooza' and eventid=(select max(eventid) from event);
```

```
select * from event
where eventname='Lollapalooza'
order by eventid;
```

| eventid | venueid | catid | dateid | eventname | starttime |
|---------|---------|-------|--------|-----------|-----------|
|---------|---------|-------|--------|-----------|-----------|

|          |     |   |      |              |                     |
|----------|-----|---|------|--------------|---------------------|
| 4274     | 102 | 9 | 1965 | Lollapalooza | 2008-05-01 19:00:00 |
| 4684     | 114 | 9 | 2105 | Lollapalooza | 2008-10-06 14:00:00 |
| 5673     | 128 | 9 | 1973 | Lollapalooza | 2008-05-01 15:00:00 |
| 5740     | 51  | 9 | 1933 | Lollapalooza | 2008-04-17 15:00:00 |
| 5856     | 119 | 9 | 1831 | Lollapalooza | 2008-01-05 14:00:00 |
| 6040     | 126 | 9 | 2145 | Lollapalooza | 2008-11-15 15:00:00 |
| 7972     | 92  | 9 | 2026 | Lollapalooza | 2008-07-19 19:30:00 |
| 8046     | 65  | 9 | 1840 | Lollapalooza | 2008-01-14 15:00:00 |
| 8518     | 48  | 9 | 1904 | Lollapalooza | 2008-03-19 15:00:00 |
| (9 rows) |     |   |      |              |                     |

Then query `stl_delete` to see the execution steps for the deletion. In this example, the query returned over 300 rows, so the output below is shortened for display purposes.

```
select query, slice, segment, step, tasknum, rows, tbl from stl_delete order by query;
```

| query | slice | segment | step | tasknum | rows | tbl    |
|-------|-------|---------|------|---------|------|--------|
| 7     | 0     | 0       | 1    | 0       | 0    | 100000 |
| 7     | 1     | 0       | 1    | 0       | 0    | 100000 |
| 8     | 0     | 0       | 1    | 2       | 0    | 100001 |
| 8     | 1     | 0       | 1    | 2       | 0    | 100001 |
| 9     | 0     | 0       | 1    | 4       | 0    | 100002 |
| 9     | 1     | 0       | 1    | 4       | 0    | 100002 |
| 10    | 0     | 0       | 1    | 6       | 0    | 100003 |
| 10    | 1     | 0       | 1    | 6       | 0    | 100003 |
| 11    | 0     | 0       | 1    | 8       | 0    | 100253 |
| 11    | 1     | 0       | 1    | 8       | 0    | 100253 |
| 12    | 0     | 0       | 1    | 0       | 0    | 100255 |
| 12    | 1     | 0       | 1    | 0       | 0    | 100255 |
| 13    | 0     | 0       | 1    | 2       | 0    | 100257 |
| 13    | 1     | 0       | 1    | 2       | 0    | 100257 |
| 14    | 0     | 0       | 1    | 4       | 0    | 100259 |
| 14    | 1     | 0       | 1    | 4       | 0    | 100259 |
| ...   |       |         |      |         |      |        |

## STL\_DISK\_FULL\_DIAG

Logs information about errors recorded when the disk is full.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                     |  |  |
|-------------|-----------|---------------------------------------------------------------------------------|--|--|
| currenttime | bigint    | The day and time the error was generated in microseconds since January 1, 2000. |  |  |
| node_num    | bigint    | The identifier for the node.                                                    |  |  |
| query_id    | bigint    | The identifier for the query that caused the error.                             |  |  |
| temp_blocks | bigint    | The number of temporary blocks created by the query.                            |  |  |

## Sample queries

The following example returns details about the data stored when there is a disk-full error.

```
select * from stl_disk_full_diag
```

The following example converts the currenttime to a timestamp.

```
select '2000-01-01'::timestamp + (currenttime/1000000.0)* interval '1 second' as
currenttime,node_num,query_id,temp_blocks from pg_catalog.stl_disk_full_diag;
```

| currenttime                | node_num | query_id | temp_blocks |
|----------------------------|----------|----------|-------------|
| 2019-05-18 19:19:18.609338 | 0        | 569399   | 70982       |
| 2019-05-18 19:37:44.755548 | 0        | 569580   | 70982       |
| 2019-05-20 13:37:20.566916 | 0        | 597424   | 70869       |

## STL\_DIST

Logs information about network activity during execution of query steps that distribute data. Network traffic is captured by numbers of rows, bytes, and packets that are sent over the network during a given step on a given slice. The duration of the step is the difference between the logged start and end times.

To identify distribution steps in a query, look for dist labels in the QUERY\_SUMMARY view or run the EXPLAIN command and then look for step attributes that include dist.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                             |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                       |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |

| Column name | Data type | Description                                          |
|-------------|-----------|------------------------------------------------------|
| rows        | bigint    | Total number of rows that were processed.            |
| bytes       | bigint    | Size, in bytes, of all the output rows for the step. |
| packets     | integer   | Total number of packets sent over the network.       |

## Sample queries

The following example returns distribution information for queries with one or more packets and duration greater than zero.

```
select query, slice, step, rows, bytes, packets,
datediff(seconds, starttime, endtime) as duration
from stl_dist
where packets>0 and datediff(seconds, starttime, endtime)>0
order by query
limit 10;
```

| query    | slice | step | rows   | bytes   | packets | duration |
|----------|-------|------|--------|---------|---------|----------|
| 567      | 1     | 4    | 49990  | 6249564 | 707     | 1        |
| 630      | 0     | 5    | 8798   | 408404  | 46      | 2        |
| 645      | 1     | 4    | 8798   | 408404  | 46      | 1        |
| 651      | 1     | 5    | 192497 | 9226320 | 1039    | 6        |
| 669      | 1     | 4    | 192497 | 9226320 | 1039    | 4        |
| 675      | 1     | 5    | 3766   | 194656  | 22      | 1        |
| 696      | 0     | 4    | 3766   | 194656  | 22      | 1        |
| 705      | 0     | 4    | 930    | 44400   | 5       | 1        |
| 111525   | 0     | 3    | 68     | 17408   | 2       | 1        |
| (9 rows) |       |      |        |         |         |          |

## STL\_ERROR

Records internal processing errors generated by the Amazon Redshift database engine. STL\_ERROR does not record SQL errors or messages. The information in STL\_ERROR is useful for troubleshooting certain errors. An AWS support engineer might ask you to provide this information as part of the troubleshooting process.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

For a list of error codes that can be generated while loading data with the Copy command, see [Load error reference \(p. 93\)](#).

## Table columns

| Column name | Data type | Description                             |
|-------------|-----------|-----------------------------------------|
| userid      | integer   | ID of the user who generated the entry. |

| Column name | Data type      | Description                                                                                                                |
|-------------|----------------|----------------------------------------------------------------------------------------------------------------------------|
| process     | character(12)  | Process that threw the exception.                                                                                          |
| recordtime  | timestamp      | Time that the error occurred.                                                                                              |
| pid         | integer        | Process ID. The <a href="#">STL_QUERY (p. 1539)</a> table contains process IDs and unique query IDs for completed queries. |
| errcode     | integer        | Error code corresponding to the error category.                                                                            |
| file        | character(90)  | Name of the source file where the error occurred.                                                                          |
| linenum     | integer        | Line number in the source file where the error occurred.                                                                   |
| context     | character(100) | Cause of the error.                                                                                                        |
| error       | character(512) | Error message.                                                                                                             |

## Sample queries

The following example retrieves the error information from STL\_ERROR.

```
select process, errcode, linenum as line,
trim(error) as err
from stl_error;

 process | errcode | line | err
-----+-----+-----+
+-----+
padbmaster | 8001 | 194 | Path prefix: s3://awssampledः/testnulls/venue.txt*
padbmaster | 8001 | 529 | Listing bucket=awssampledः prefix=tests/category-csv-
quotes
padbmaster | 2 | 190 | database "template0" is not currently accepting
connections
padbmaster | 32 | 1956 | pq_flush: could not send data to client: Broken pipe
(4 rows)
```

## STL\_EXPLAIN

Displays the EXPLAIN plan for a query that has been submitted for execution.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type | Description                                                                                 |
|-------------|-----------|---------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                     |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.               |
| nodeid      | integer   | Plan node identifier, where a node maps to one or more steps in the execution of the query. |

| Column name | Data type      | Description                                                                                                                                                      |
|-------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| parentid    | integer        | Plan node identifier for a parent node. A parent node has some number of child nodes. For example, a merge join is the parent of the scans on the joined tables. |
| plannode    | character(400) | The node text from the EXPLAIN output. Plan nodes that refer to execution on compute nodes are prefixed with <b>XN</b> in the EXPLAIN output.                    |
| info        | character(400) | Qualifier and filter information for the plan node. For example, join conditions and WHERE clause restrictions are included in this column.                      |

## Sample queries

Consider the following EXPLAIN output for an aggregate join query:

```
explain select avg(datediff(day, listtime, saletime)) as avgwait
from sales, listing where sales.listid = listing.listid;
 QUERY PLAN

XN Aggregate (cost=6350.30..6350.31 rows=1 width=16)
 -> XN Hash Join DS_DIST_NONE (cost=47.08..6340.89 rows=3766 width=16)
 Hash Cond: ("outer".listid = "inner".listid)
 -> XN Seq Scan on listing (cost=0.00..1924.97 rows=192497 width=12)
 -> XN Hash (cost=37.66..37.66 rows=3766 width=12)
 -> XN Seq Scan on sales (cost=0.00..37.66 rows=3766 width=12)
(6 rows)
```

If you run this query and its query ID is 10, you can use the **STL\_EXPLAIN** table to see the same kind of information that the EXPLAIN command returns:

```
select query,nodeid,parentid,substring(plannode from 1 for 30),
substring(info from 1 for 20) from stl_explain
where query=10 order by 1,2;

query| nodeid |parentid| substring | substring
-----+-----+-----+-----+-----+
10 | 1 | 0 | XN Aggregate (cost=6717.61..6 |
10 | 2 | 1 | -> XN Merge Join DS_DIST_NO | Merge Cond:("outer"
10 | 3 | 2 | -> XN Seq Scan on lis |
10 | 4 | 2 | -> XN Seq Scan on sal |
(4 rows)
```

Consider the following query:

```
select event.eventid, sum(pricepaid)
from event, sales
where event.eventid=sales.eventid
group by event.eventid order by 2 desc;

eventid | sum
-----+-----
 289 | 51846.00
 7895 | 51049.00
 1602 | 50301.00
 851 | 49956.00
```

```
7315 | 49823.00
...
```

If this query's ID is 15, the following system view query returns the plan nodes that were completed. In this case, the order of the nodes is reversed to show the actual order of execution:

```
select query,nodeid,parentid,substring(plannode from 1 for 56)
from stl_explain where query=15 order by 1, 2 desc;

query|nodeid|parentid| substring
-----+-----+-----+
15 | 8 | 7 | -> XN Seq Scan on eve
15 | 7 | 5 | -> XN Hash(cost=87.98..87.9
15 | 6 | 5 | -> XN Seq Scan on sales(cost
15 | 5 | 4 | -> XN Hash Join DS_DIST_OUTER(cost
15 | 4 | 3 | -> XN HashAggregate(cost=862286577.07..
15 | 3 | 2 | -> XN Sort(cost=1000862287175.47..10008622871
15 | 2 | 1 | -> XN Network(cost=1000862287175.47..1000862287197.
15 | 1 | 0 | XN Merge(cost=1000862287175.47..1000862287197.46 rows=87
(8 rows)
```

The following query retrieves the query IDs for any query plans that contain a window function:

```
select query, trim(plannode) from stl_explain
where plannode like '%Window%';

query| btrim
-----+
26 | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
27 | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
(2 rows)
```

## STL\_FILE\_SCAN

Returns the files that Amazon Redshift read while loading data by using the COPY command.

Querying this view can help troubleshoot data load errors. STL\_FILE\_SCAN can be particularly helpful with pinpointing issues in parallel data loads, because parallel data loads typically load many files with a single COPY command.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type     | Description                                                                   |
|-------------|---------------|-------------------------------------------------------------------------------|
| userid      | integer       | ID of the user who generated the entry.                                       |
| query       | integer       | Query ID. The query column can be used to join other system tables and views. |
| slice       | integer       | Number that identifies the slice where the query was running.                 |
| name        | character(90) | Full path and name of the file that was loaded.                               |
| lines       | bigint        | Number of lines read from the file.                                           |
| bytes       | bigint        | Number of bytes read from the file.                                           |

| Column name  | Data type | Description                                                                                                                                                   |
|--------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| loadtime     | bigint    | Amount of time spent loading the file (in microseconds).                                                                                                      |
| curtime      | Timestamp | Timestamp representing the time that Amazon Redshift started processing the file.                                                                             |
| is_partial   | integer   | Value that if true (1) indicates the input file is split into ranges during a COPY operation. If this value is false (0), the input file isn't split.         |
| start_offset | bigint    | Value that, if the input file is split during a COPY operation, indicates the offset value of the split (in bytes). If the file isn't split, this value is 0. |

## Sample queries

The following query retrieves the names and load times of any files that took over 1,000,000 microseconds for Amazon Redshift to read.

```
select trim(name)as name, loadtime from stl_file_scan
where loadtime > 1000000;
```

This query returns the following example output.

| name                      | loadtime |
|---------------------------|----------|
| listings_pipe.txt         | 9458354  |
| allusers_pipe.txt         | 2963761  |
| allevents_pipe.txt        | 1409135  |
| tickit/listings_pipe.txt  | 7071087  |
| tickit/allevents_pipe.txt | 1237364  |
| tickit/allusers_pipe.txt  | 2535138  |
| listings_pipe.txt         | 6706370  |
| allusers_pipe.txt         | 3579461  |
| allevents_pipe.txt        | 1313195  |
| tickit/allusers_pipe.txt  | 3236060  |
| tickit/listings_pipe.txt  | 4980108  |

(11 rows)

## STL\_HASH

Analyzes hash execution steps for queries.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                   |
|-------------|-----------|-------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                       |
| query       | integer   | Query ID. The query column can be used to join other system tables and views. |

| Column name             | Data type    | Description                                                                                                                                                                         |
|-------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| slice                   | integer      | Number that identifies the slice where the query was running.                                                                                                                       |
| segment                 | integer      | Number that identifies the query segment.                                                                                                                                           |
| step                    | integer      | Query step that ran.                                                                                                                                                                |
| starttime               | timestamp    | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime                 | timestamp    | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum                 | integer      | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows                    | bigint       | Total number of rows that were processed.                                                                                                                                           |
| bytes                   | bigint       | Size, in bytes, of all the output rows for the step.                                                                                                                                |
| slots                   | integer      | Total number of hash buckets.                                                                                                                                                       |
| occupied                | integer      | Total number of slots that contain records.                                                                                                                                         |
| maxlength               | integer      | Size of the largest slot.                                                                                                                                                           |
| tbl                     | integer      | Table ID.                                                                                                                                                                           |
| is_diskbased            | character(1) | If true (t), the query was performed as a disk-based operation. If false (f), the query was performed in memory.                                                                    |
| workmem                 | bigint       | Total number of bytes of working memory assigned to the step.                                                                                                                       |
| num_parts               | integer      | Total number of partitions that a hash table was divided into during a hash step.                                                                                                   |
| est_rows                | bigint       | Estimated number of rows to be hashed.                                                                                                                                              |
| num_blocks_permitted    | integer      | This information is for internal use only.                                                                                                                                          |
| resizes                 | integer      | This information is for internal use only.                                                                                                                                          |
| checksum                | bigint       | This information is for internal use only.                                                                                                                                          |
| runtime_filter_size     | integer      | Size of the runtime filter in bytes.                                                                                                                                                |
| max_runtime_filter_size | integer      | Maximum size of the runtime filter in bytes.                                                                                                                                        |

## Sample queries

The following example returns information about the number of partitions that were used in a hash for query 720, and indicates that none of the steps ran on disk.

```
select slice, rows, bytes, occupied, workmem, num_parts, est_rows, num_blocks_permitted,
 is_diskbased
 from stl_hash
```

```
where query=720 and segment=5
order by slice;
```

| slice   rows   bytes   occupied   workmem   num_parts   est_rows   num_blocks_permitted<br>  is_diskbased |                                                  |  |  |  |  |  |  |
|-----------------------------------------------------------------------------------------------------------|--------------------------------------------------|--|--|--|--|--|--|
| -----+-----+-----+-----+-----+-----+-----+-----+                                                          | -----+-----+-----+-----+-----+-----+-----+-----+ |  |  |  |  |  |  |
| 0   145   585800   f                                                                                      | 1   88866816   16   1   52                       |  |  |  |  |  |  |
| 1   0   0   f                                                                                             | 0   0   16   1   52                              |  |  |  |  |  |  |
| (2 rows)                                                                                                  |                                                  |  |  |  |  |  |  |

## STL\_HASHJOIN

Analyzes hash join execution steps for queries.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                             |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                       |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows        | bigint    | Total number of rows that were processed.                                                                                                                                           |
| tbl         | integer   | Table ID.                                                                                                                                                                           |
| num_parts   | integer   | Total number of partitions that a hash table was divided into during a hash step.                                                                                                   |
| join_type   | integer   | The type of join for the step: <ul style="list-style-type: none"> <li>• 0. The query used an inner join.</li> <li>• 1. The query used a left outer join.</li> </ul>                 |

| Column name      | Data type    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                  |              | <ul style="list-style-type: none"> <li>• 2. The query used a full outer join.</li> <li>• 3. The query used a right outer join.</li> <li>• 4. The query used a UNION operator.</li> <li>• 5. The query used an IN condition.</li> <li>• 6. This information is for internal use only.</li> <li>• 7. This information is for internal use only.</li> <li>• 8. This information is for internal use only.</li> <li>• 9. This information is for internal use only.</li> <li>• 10. This information is for internal use only.</li> <li>• 11. This information is for internal use only.</li> <li>• 12. This information is for internal use only.</li> </ul> |
| hash_looped      | character(1) | This information is for internal use only.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| switched_parts   | character(1) | Indicates whether the build (or outer) and probe (or inner) sides have switched.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| used_prefetch_id | character(1) | This information is for internal use only.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| hash_segment     | integer      | The segment of the corresponding hash step.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| hash_step        | integer      | The step number of the corresponding hash step.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| checksum         | bigint       | This information is for internal use only.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| distribution     | integer      | This information is for internal use only.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

## Sample queries

The following example returns the number of partitions used in a hash join for query 720.

```
select query, slice, tbl, num_parts
from stl_hashjoin
where query=720 limit 10;
```

| query    | slice | tbl | num_parts |
|----------|-------|-----|-----------|
| 720      | 0     | 243 | 1         |
| 720      | 1     | 243 | 1         |
| (2 rows) |       |     |           |

## STL\_INSERT

Analyzes insert execution steps for queries.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                             |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                       |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows        | bigint    | Total number of rows that were processed.                                                                                                                                           |
| tbl         | integer   | Table ID.                                                                                                                                                                           |

## Sample queries

The following example returns insert execution steps for the most recent query.

```
select slice, segment, step, tasknum, rows, tbl
from stl_insert
where query=pg_last_query_id();
```

| slice    | segment | step | tasknum | rows  | tbl    |
|----------|---------|------|---------|-------|--------|
| 0        | 2       | 2    | 15      | 24958 | 100548 |
| 1        | 2       | 2    | 15      | 25032 | 100548 |
| (2 rows) |         |      |         |       |        |

## STL\_LIMIT

Analyzes the execution steps that occur when a LIMIT clause is used in a SELECT query.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                             |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                       |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows        | bigint    | Total number of rows that were processed.                                                                                                                                           |
| checksum    | bigint    | This information is for internal use only.                                                                                                                                          |

## Sample queries

In order to generate a row in STL\_LIMIT, this example first runs the following query against the VENUE table using the LIMIT clause.

```
select * from venue
order by 1
limit 10;
```

| venueid   | venuename                  | venuecity       | venuestate | venueseats |
|-----------|----------------------------|-----------------|------------|------------|
| 1         | Toyota Park                | Bridgeview      | IL         | 0          |
| 2         | Columbus Crew Stadium      | Columbus        | OH         | 0          |
| 3         | RFK Stadium                | Washington      | DC         | 0          |
| 4         | CommunityAmerica Ballpark  | Kansas City     | KS         | 0          |
| 5         | Gillette Stadium           | Foxborough      | MA         | 68756      |
| 6         | New York Giants Stadium    | East Rutherford | NJ         | 80242      |
| 7         | BMO Field                  | Toronto         | ON         | 0          |
| 8         | The Home Depot Center      | Carson          | CA         | 0          |
| 9         | Dick's Sporting Goods Park | Commerce City   | CO         | 0          |
| 10        | Pizza Hut Park             | Frisco          | TX         | 0          |
| (10 rows) |                            |                 |            |            |

Next, run the following query to find the query ID of the last query you ran against the VENUE table.

```
select max(query)
from stl_query;
```

```
max

127128
(1 row)
```

Optionally, you can run the following query to verify that the query ID corresponds to the LIMIT query you previously ran.

```
select query, trim(querytxt)
from stl_query
where query=127128;
```

```
query | btrim

127128 | select * from venue order by 1 limit 10;
(1 row)
```

Finally, run the following query to return information about the LIMIT query from the STL\_LIMIT table.

```
select slice, segment, step, starttime, endtime, tasknum
from stl_limit
where query=127128
order by starttime, endtime;
```

```
slice | segment | step | starttime | endtime |
tasknum
-----+-----+-----+-----+-----+
15 | 1 | 3 | 2013-09-06 22:56:43.608114 | 2013-09-06 22:56:43.609383 |
15 | 0 | 3 | 2013-09-06 22:56:43.608708 | 2013-09-06 22:56:43.609521 |
10000 | 2 | 2 | 2013-09-06 22:56:43.612506 | 2013-09-06 22:56:43.612668 |
0 | 0 | 0 | | |
(3 rows)
```

## STL\_LOAD\_COMMITS

Returns information to track or troubleshoot a data load.

This view records the progress of each data file as it is loaded into a database table.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type      | Description                                                                   |
|-------------|----------------|-------------------------------------------------------------------------------|
| userid      | integer        | ID of the user who generated the entry.                                       |
| query       | integer        | Query ID. The query column can be used to join other system tables and views. |
| slice       | integer        | Slice loaded for this entry.                                                  |
| name        | character(256) | System-defined value.                                                         |

| Column name   | Data type      | Description                                                                                                                                                                                                                                             |
|---------------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| filename      | character(256) | Name of file being tracked.                                                                                                                                                                                                                             |
| byte_offset   | integer        | This information is for internal use only.                                                                                                                                                                                                              |
| lines_scanned | integer        | Number of lines scanned from the load file. This number may not match the number of rows that are actually loaded. For example, the load may scan but tolerate a number of bad records, based on the MAXERROR option in the COPY command.               |
| errors        | integer        | This information is for internal use only.                                                                                                                                                                                                              |
| curtime       | timestamp      | Time that this entry was last updated.                                                                                                                                                                                                                  |
| status        | integer        | This information is for internal use only.                                                                                                                                                                                                              |
| file_format   | character(16)  | Format of the load file. Possible values are as follows: <ul style="list-style-type: none"> <li>• Avro</li> <li>• JSON</li> <li>• ORC</li> <li>• Parquet</li> <li>• Text</li> </ul>                                                                     |
| is_partial    | integer        | Value that if true (1) indicates the input file is split into ranges during a COPY operation. If this value is false (0), the input file isn't split.                                                                                                   |
| start_offset  | bigint         | Value that, if the input file is split during a COPY operation, indicates the offset value of the split (in bytes). Each file split is logged as a separate record with the corresponding start_offset value. If the file isn't split, this value is 0. |
| copy_job_id   | bigint         | The copy job identifier. A 0 indicates no job identifier.                                                                                                                                                                                               |

## Sample queries

The following example returns details for the last COPY operation.

```
select query, trim(filename) as file, curtime as updated
from stl_load_commits
where query = pg_last_copy_id();

query | file | updated
-----+-----+-----+
 28554 | s3://dw-ticket/category_pipe.txt | 2013-11-01 17:14:52.648486
(1 row)
```

The following query contains entries for a fresh load of the tables in the TICKIT database:

```
select query, trim(filename), curtime
from stl_load_commits
where filename like '%ticket%' order by query;
```

|                         |
|-------------------------|
| query   btrim   curtime |
|-------------------------|

```

-----+-----+-----+
22475 | tickit/allusers_pipe.txt | 2013-02-08 20:58:23.274186
22478 | tickit/venue_pipe.txt | 2013-02-08 20:58:25.070604
22480 | tickit/category_pipe.txt | 2013-02-08 20:58:27.333472
22482 | tickit/date2008_pipe.txt | 2013-02-08 20:58:28.608305
22485 | tickit/allevents_pipe.txt | 2013-02-08 20:58:29.99489
22487 | tickit/listings_pipe.txt | 2013-02-08 20:58:37.632939
22593 | tickit/allusers_pipe.txt | 2013-02-08 21:04:08.400491
22596 | tickit/venue_pipe.txt | 2013-02-08 21:04:10.056055
22598 | tickit/category_pipe.txt | 2013-02-08 21:04:11.465049
22600 | tickit/date2008_pipe.txt | 2013-02-08 21:04:12.461502
22603 | tickit/allevents_pipe.txt | 2013-02-08 21:04:14.785124
22605 | tickit/listings_pipe.txt | 2013-02-08 21:04:20.170594

```

(12 rows)

The fact that a record is written to the log file for this system view does not mean that the load committed successfully as part of its containing transaction. To verify load commits, query the `STL.utilitytext` view and look for the `COMMIT` record that corresponds with a `COPY` transaction. For example, this query joins `STL.load_commits` and `STL.query` based on a subquery against `STL.utilitytext`:

```

select l.query,rtrim(l.filename),q.xid
from stl_load_commits l, stl_query q
where l.query=q.query
and exists
(select xid from stl_utilitytext where xid=q.xid and rtrim("text")='COMMIT');

query | rtrim | xid
-----+-----+-----+
22600 | tickit/date2008_pipe.txt | 68311
22480 | tickit/category_pipe.txt | 68066
7508 | allusers_pipe.txt | 23365
7552 | category_pipe.txt | 23415
7576 | allevents_pipe.txt | 23429
7516 | venue_pipe.txt | 23390
7604 | listings_pipe.txt | 23445
22596 | tickit/venue_pipe.txt | 68309
22605 | tickit/listings_pipe.txt | 68316
22593 | tickit/allusers_pipe.txt | 68305
22485 | tickit/allevents_pipe.txt | 68071
7561 | allevents_pipe.txt | 23429
7541 | category_pipe.txt | 23415
7558 | date2008_pipe.txt | 23428
22478 | tickit/venue_pipe.txt | 68065
526 | date2008_pipe.txt | 2572
7466 | allusers_pipe.txt | 23365
22482 | tickit/date2008_pipe.txt | 68067
22598 | tickit/category_pipe.txt | 68310
22603 | tickit/allevents_pipe.txt | 68315
22475 | tickit/allusers_pipe.txt | 68061
547 | date2008_pipe.txt | 2572
22487 | tickit/listings_pipe.txt | 68072
7531 | venue_pipe.txt | 23390
7583 | listings_pipe.txt | 23445

```

(25 rows)

The following examples highlight `is_partial` and `start_offset` column values.

```

-- Single large file copy without scan range
SELECT count(*) FROM stl_load_commits WHERE query = pg_last_copy_id();
1

```

```
-- Single large uncompressed, delimited file copy with scan range
SELECT count(*) FROM stl_load_commits WHERE query = pg_last_copy_id();
16

-- Scan range offset logging in the file at 64MB boundary.
SELECT start_offset FROM stl_load_commits
WHERE query = pg_last_copy_id() ORDER BY start_offset;
0
67108864
134217728
201326592
268435456
335544320
402653184
469762048
536870912
603979776
671088640
738197504
805306368
872415232
939524096
1006632960
```

## STL\_LOAD\_ERRORS

Displays the records of all Amazon Redshift load errors.

STL\_LOAD\_ERRORS contains a history of all Amazon Redshift load errors. See [Load error reference \(p. 93\)](#) for a comprehensive list of possible load errors and explanations.

Query [STL\\_LOADERROR\\_DETAIL \(p. 1528\)](#) for additional details, such as the exact data row and column where a parse error occurred, after you query STL\_LOAD\_ERRORS to find out general information about the error.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type      | Description                                                                                                                          |
|-------------|----------------|--------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer        | ID of the user who generated the entry.                                                                                              |
| slice       | integer        | Slice where the error occurred.                                                                                                      |
| tbl         | integer        | Table ID.                                                                                                                            |
| starttime   | timestamp      | Start time in UTC for the load.                                                                                                      |
| session     | integer        | Session ID for the session performing the load.                                                                                      |
| query       | integer        | Query ID. The query column can be used to join other system tables and views.                                                        |
| filename    | character(256) | Complete path to the input file for the load.                                                                                        |
| line_number | bigint         | Line number in the load file with the error. For COPY from JSON, the line number of the last line of the JSON object with the error. |

| Column name              | Data type       | Description                                                                                                                                                                                                                     |
|--------------------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| colname                  | character(127)  | Field with the error.                                                                                                                                                                                                           |
| type                     | character(10)   | Data type of the field.                                                                                                                                                                                                         |
| col_length               | character(10)   | Column length, if applicable. This field is populated when the data type has a limit length. For example, for a column with a data type of "character(3)", this column will contain the value "3".                              |
| position                 | integer         | Position of the error in the field.                                                                                                                                                                                             |
| raw_line                 | character(1024) | Raw load data that contains the error. Multibyte characters in the load data are replaced with a period.                                                                                                                        |
| raw_field_valuehar(1024) |                 | The pre-parsing value for the field "colname" that lead to the parsing error.                                                                                                                                                   |
| err_code                 | integer         | Error code.                                                                                                                                                                                                                     |
| err_reason               | character(100)  | Explanation for the error.                                                                                                                                                                                                      |
| is_partial               | integer         | Value that if true (1) indicates the input file is split into ranges during a COPY operation. If this value is false (0), the input file isn't split.                                                                           |
| start_offset             | bigint          | Value that, if the input file is split during a COPY operation, indicates the offset value of the split (in bytes). If the line number in the file is unknown, the line number is -1. If the file isn't split, this value is 0. |
| copy_job_id              | bigint          | The copy job identifier. A 0 indicates no job identifier.                                                                                                                                                                       |

## Sample queries

The following query joins STL\_LOAD\_ERRORS to STL\_LOADERROR\_DETAIL to view the details errors that occurred during the most recent load.

```
select d.query, substring(d.filename,14,20),
d.line_number as line,
substring(d.value,1,16) as value,
substring(le.err_reason,1,48) as err_reason
from stl_loaderror_detail d, stl_load_errors le
where d.query = le.query
and d.query = pg_last_copy_id();

query | substring | line | value | err_reason
-----+-----+-----+-----+
 558| allusers_pipe.txt | 251 | 251 | String contains invalid or
 unsupported UTF8 code
 558| allusers_pipe.txt | 251 | ZRU29FGR | String contains invalid or
 unsupported UTF8 code
 558| allusers_pipe.txt | 251 | Kaitlin | String contains invalid or
 unsupported UTF8 code
 558| allusers_pipe.txt | 251 | Walter | String contains invalid or
 unsupported UTF8 code
```

The following example uses `STL_LOAD_ERRORS` with `STV_TBL_PERM` to create a new view, and then uses that view to determine what errors occurred while loading data into the `EVENT` table:

```
create view loadview as
(select distinct tbl, trim(name) as table_name, query, starttime,
trim(filename) as input, line_number, colname, err_code,
trim(err_reason) as reason
from stl_load_errors sl, stv_tbl_perm sp
where sl.tbl = sp.id);
```

Next, the following query actually returns the last error that occurred while loading the `EVENT` table:

```
select table_name, query, line_number, colname, starttime,
trim(reason) as error
from loadview
where table_name = 'event'
order by line_number limit 1;
```

The query returns the last load error that occurred for the `EVENT` table. If no load errors occurred, the query returns zero rows. In this example, the query returns a single error:

| table_name | query                                                                | line_number | colname | error | starttime           |
|------------|----------------------------------------------------------------------|-------------|---------|-------|---------------------|
| event      | 309   0   5   Error in Timestamp value or format [%Y-%m-%d %H:%M:%S] |             |         |       | 2014-04-22 15:12:44 |

(1 row)

In cases where the `COPY` command automatically splits large, uncompressed, text-delimited file data to facilitate parallelism, the `line_number`, `is_partial`, and `start_offset` columns show information pertaining to splits. (The line number can be unknown in cases where the line number from the original file is unavailable.)

```
--scan ranges information
SELECT line_number, POSITION, btrim(raw_line), btrim(raw_field_value),
btrim(err_reason), is_partial, start_offset FROM stl_load_errors
WHERE query = pg_last_copy_id();

--result
-1,51,"1008771|13463413|463414|2|28.00|38520.72|0.06|0.07|NO|1998-08-30|1998-09-25
1998-09-04|TAKE BACK RETURN|RAIL|ans cajole sly","NO","Char length exceeds DDL
length",1,67108864
```

## STL\_LOADERROR\_DETAIL

Displays a log of data parse errors that occurred while using a `COPY` command to load tables. To conserve disk space, a maximum of 20 errors per node slice are logged for each load operation.

A parse error occurs when Amazon Redshift cannot parse a field in a data row while loading it into a table. For example, if a table column is expecting an integer data type and the data file contains a string of letters in that field, it causes a parse error.

Query `STL_LOADERROR_DETAIL` for additional details, such as the exact data row and column where a parse error occurred, after you query [STL\\_LOAD\\_ERRORS \(p. 1526\)](#) to find out general information about the error.

The `STL_LOADERROR_DETAIL` view contains all data columns including and prior to the column where the parse error occurred. Use the `VALUE` field to see the data value that was actually parsed in this column, including the columns that parsed correctly up to the error.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name              | Data type                    | Description                                                                                                                                                                                        |
|--------------------------|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>userid</code>      | <code>integer</code>         | ID of the user who generated the entry.                                                                                                                                                            |
| <code>slice</code>       | <code>integer</code>         | Slice where the error occurred.                                                                                                                                                                    |
| <code>session</code>     | <code>integer</code>         | Session ID for the session performing the load.                                                                                                                                                    |
| <code>query</code>       | <code>integer</code>         | Query ID. The query column can be used to join other system tables and views.                                                                                                                      |
| <code>filename</code>    | <code>character(256)</code>  | Complete path to the input file for the load.                                                                                                                                                      |
| <code>line_number</code> | <code>bigint</code>          | Line number in the load file with the error.                                                                                                                                                       |
| <code>field</code>       | <code>integer</code>         | Field with the error.                                                                                                                                                                              |
| <code>colname</code>     | <code>character(1024)</code> | Column name.                                                                                                                                                                                       |
| <code>value</code>       | <code>character(1024)</code> | Parsed data value of the field. (May be truncated.) Multibyte characters in the load data are replaced with a period.                                                                              |
| <code>is_null</code>     | <code>integer</code>         | Whether or not the parsed value is null.                                                                                                                                                           |
| <code>type</code>        | <code>character(10)</code>   | Data type of the field.                                                                                                                                                                            |
| <code>col_length</code>  | <code>character(10)</code>   | Column length, if applicable. This field is populated when the data type has a limit length. For example, for a column with a data type of "character(3)", this column will contain the value "3". |

## Sample query

The following query joins `STL_LOAD_ERRORS` to `STL_LOADERROR_DETAIL` to view the details of a parse error that occurred while loading the `EVENT` table, which has a table ID of 100133:

```
select d.query, d.line_number, d.value,
le.raw_line, le.err_reason
from stl_loaderror_detail d, stl_load_errors le
where
d.query = le.query
and tbl = 100133;
```

The following sample output shows the columns that loaded successfully, including the column with the error. In this example, two columns successfully loaded before the parse error occurred in the third column, where a character string was incorrectly parsed for a field expecting an integer. Because the field expected an integer, it parsed the string "aaa", which is uninitialized data, as a null and generated a parse error. The output shows the raw value, parsed value, and error reason:

| query    | line_number | value | raw_line | err_reason    |
|----------|-------------|-------|----------|---------------|
| 4        | 3           | 1201  | 1201     | Invalid digit |
| 4        | 3           | 126   | 126      | Invalid digit |
| 4        | 3           |       | aaa      | Invalid digit |
| (3 rows) |             |       |          |               |

When a query joins STL\_LOAD\_ERRORS and STL\_LOADERROR\_DETAIL, it displays an error reason for each column in the data row, which simply means that an error occurred in that row. The last row in the results is the actual column where the parse error occurred.

## STL\_MERGE

Analyzes merge execution steps for queries. These steps occur when the results of parallel operations (such as sorts and joins) are merged for subsequent processing.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                             |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                       |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution, with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution, with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows        | bigint    | Total number of rows that were processed.                                                                                                                                           |

### Sample queries

The following example returns 10 merge execution results.

```
select query, step, starttime, endtime, tasknum, rows
from stl_merge
limit 10;
```

|       |      |           |         |         |      |
|-------|------|-----------|---------|---------|------|
| query | step | starttime | endtime | tasknum | rows |
|-------|------|-----------|---------|---------|------|

|           |   |                     |                     |   |   |
|-----------|---|---------------------|---------------------|---|---|
| 9         | 0 | 2013-08-12 20:08:14 | 2013-08-12 20:08:14 | 0 | 0 |
| 12        | 0 | 2013-08-12 20:09:10 | 2013-08-12 20:09:10 | 0 | 0 |
| 15        | 0 | 2013-08-12 20:10:24 | 2013-08-12 20:10:24 | 0 | 0 |
| 20        | 0 | 2013-08-12 20:11:27 | 2013-08-12 20:11:27 | 0 | 0 |
| 26        | 0 | 2013-08-12 20:12:28 | 2013-08-12 20:12:28 | 0 | 0 |
| 32        | 0 | 2013-08-12 20:14:33 | 2013-08-12 20:14:33 | 0 | 0 |
| 38        | 0 | 2013-08-12 20:16:43 | 2013-08-12 20:16:43 | 0 | 0 |
| 44        | 0 | 2013-08-12 20:17:05 | 2013-08-12 20:17:05 | 0 | 0 |
| 50        | 0 | 2013-08-12 20:18:48 | 2013-08-12 20:18:48 | 0 | 0 |
| 56        | 0 | 2013-08-12 20:20:48 | 2013-08-12 20:20:48 | 0 | 0 |
| (10 rows) |   |                     |                     |   |   |

## STL\_MERGEJOIN

Analyzes merge join execution steps for queries.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                             |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                       |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows        | bigint    | Total number of rows that were processed.                                                                                                                                           |
| tbl         | integer   | Table ID. This is the ID for the inner table that was used in the merge join.                                                                                                       |
| checksum    | bigint    | This information is for internal use only.                                                                                                                                          |

### Sample queries

The following example returns merge join results for the most recent query.

```
select sum(s.qtysold), e.eventname
```

```
from event e, listing l, sales s
where e.eventid=l.eventid
and l.listid= s.listid
group by e.eventname;

select * from stl_mergejoin where query=pg_last_query_id();
```

| userid  | query | slice | segment | step | starttime           | endtime             |
|---------|-------|-------|---------|------|---------------------|---------------------|
| tasknum | rows  | tbl   |         |      |                     |                     |
| 100     | 27399 | 3     |         | 4    | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
| 19      | 43428 | 240   |         |      |                     |                     |
| 100     | 27399 | 0     |         | 4    | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
| 19      | 43159 | 240   |         |      |                     |                     |
| 100     | 27399 | 2     |         | 4    | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
| 19      | 42778 | 240   |         |      |                     |                     |
| 100     | 27399 | 1     |         | 4    | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
| 19      | 43091 | 240   |         |      |                     |                     |

## STL\_MV\_STATE

The STL\_MV\_STATE view contains a row for every state transition of a materialized view.

For more information about materialized views, see [Creating materialized views in Amazon Redshift \(p. 198\)](#).

STL\_MV\_STATE is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name       | Data type | Description                                                                                                                                                                                                                                                                                                                        |
|-------------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid            | bigint    | The ID of the user who created the event.                                                                                                                                                                                                                                                                                          |
| starttime         | timestamp | The start time of the event.                                                                                                                                                                                                                                                                                                       |
| xid               | bigint    | The transaction id of the event.                                                                                                                                                                                                                                                                                                   |
| event_desc        | char(500) | The event that prompted the state change. Example values include the following: <ul style="list-style-type: none"> <li>• Column type was changed</li> <li>• Column was dropped</li> <li>• Column was renamed</li> <li>• Schema name was changed</li> <li>• Small table conversion</li> <li>• TRUNCATE</li> <li>• Vacuum</li> </ul> |
| db_name           | char(128) | The database that contains the materialized view.                                                                                                                                                                                                                                                                                  |
| base_table_schema | char(128) | The schema of the base table.                                                                                                                                                                                                                                                                                                      |
| base_table_name   | char(128) | The name of the base table.                                                                                                                                                                                                                                                                                                        |

| Column name | Data type     | Description                                                                                                                                |
|-------------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| mv_schema   | char(128)     | The schema of the materialized view.                                                                                                       |
| mv_name     | char(128)     | The name of the materialized view.                                                                                                         |
| state       | character(32) | The changed state of the materialized view as follows: <ul style="list-style-type: none"><li>• Recompute</li><li>• Unrefreshable</li></ul> |

The following table shows example combinations of event\_desc and state.

| event_desc              | state         |
|-------------------------|---------------|
| TRUNCATE                | Recompute     |
| TRUNCATE                | Recompute     |
| Small table conversion  | Recompute     |
| Vacuum                  | Recompute     |
| Column was renamed      | Unrefreshable |
| Column was dropped      | Unrefreshable |
| Table was renamed       | Unrefreshable |
| Column type was changed | Unrefreshable |
| Schema name was changed | Unrefreshable |

## Sample query

To view the log of state transitions of materialized views, run the following query.

```
select * from stl_mv_state;
```

This query returns the following sample output:

| userid            | starttime                  | xid       | event_desc                  | db_name |
|-------------------|----------------------------|-----------|-----------------------------|---------|
| base_table_schema | base_table_name            | mv_schema | mv_name                     |         |
| state             |                            |           |                             |         |
| 138               | 2020-02-14 02:21:25.578885 | 5180      | TRUNCATE                    | dev     |
| public            | mv_base_table              | public    | mv_test                     |         |
| Recompute         |                            |           |                             |         |
| 138               | 2020-02-14 02:21:56.846774 | 5275      | Column was dropped          | dev     |
|                   | mv_base_table              | public    | mv_test                     |         |
| Unrefreshable     |                            |           |                             |         |
| 100               | 2020-02-13 22:09:53.041228 | 1794      | Column was renamed          | dev     |
|                   | mv_base_table              | public    | mv_test                     |         |
| Unrefreshable     |                            |           |                             |         |
| 1                 | 2020-02-13 22:10:23.630914 | 1893      | ALTER TABLE ALTER SORTKEY   | dev     |
| public            | mv_base_table_sorted       | public    | mv_test                     |         |
| Recompute         |                            |           |                             |         |
| 1                 | 2020-02-17 22:57:22.497989 | 8455      | ALTER TABLE ALTER DISTSTYLE | dev     |
| public            | mv_base_table              | public    | mv_test                     |         |
| Recompute         |                            |           |                             |         |
| 173               | 2020-02-17 22:57:23.591434 | 8504      | Table was renamed           | dev     |
|                   | mv_base_table              | public    | mv_test                     |         |
| Unrefreshable     |                            |           |                             |         |

|                      |                            |      |                           |                      |         |  |
|----------------------|----------------------------|------|---------------------------|----------------------|---------|--|
| 173                  | 2020-02-17 22:57:27.229423 | 8592 | Column type was changed   | dev                  |         |  |
|                      |                            |      | mv_base_table             | public               | mv_test |  |
| <b>Unrefreshable</b> |                            |      |                           |                      |         |  |
| 197                  | 2020-02-17 22:59:06.212569 | 9668 | TRUNCATE                  | dev                  |         |  |
|                      |                            |      | mv_base_table             | schemaf796e415850f4f | mv_test |  |
| <b>Recompute</b>     |                            |      |                           |                      |         |  |
| 138                  | 2020-02-14 02:21:55.705655 | 5226 | Column was renamed        | dev                  |         |  |
|                      |                            |      | mv_base_table             | public               | mv_test |  |
| <b>Unrefreshable</b> |                            |      |                           |                      |         |  |
| 1                    | 2020-02-14 02:22:26.292434 | 5325 | ALTER TABLE ALTER SORTKEY | dev                  |         |  |
| public               |                            |      | mv_base_table_sorted      | public               | mv_test |  |
| <b>Recompute</b>     |                            |      |                           |                      |         |  |

## STL\_NESTLOOP

Analyzes nested-loop join execution steps for queries.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                             |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                       |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows        | bigint    | Total number of rows that were processed.                                                                                                                                           |
| tbl         | integer   | Table ID.                                                                                                                                                                           |
| checksum    | bigint    | This information is for internal use only.                                                                                                                                          |

### Sample queries

Because the following query neglects to join the CATEGORY table, it produces a partial Cartesian product, which is not recommended. It is shown here to illustrate a nested loop.

```
select count(event.eventname), event.eventname, category.catname, date.caldate
from event, category, date
where event.dateid = date.dateid
group by event.eventname, category.catname, date.caldate;
```

The following query shows the results from the previous query in the STL\_NESTLOOP view.

```
select query, slice, segment as seg, step,
datediff(msc, starttime, endtime) as duration, tasknum, rows, tbl
from stl_nestloop
where query = pg_last_query_id();
```

| query | slice | seg | step | duration | tasknum | rows  | tbl |
|-------|-------|-----|------|----------|---------|-------|-----|
| 6028  | 0     | 4   | 5    | 41       | 22      | 24277 | 240 |
| 6028  | 1     | 4   | 5    | 26       | 23      | 24189 | 240 |
| 6028  | 3     | 4   | 5    | 25       | 23      | 24376 | 240 |
| 6028  | 2     | 4   | 5    | 54       | 22      | 23936 | 240 |

## STL\_PARSE

Analyzes query steps that parse strings into binary values for loading.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                             |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                       |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows        | bigint    | Total number of rows that were processed.                                                                                                                                           |

## Sample queries

The following example returns all query step results for slice 1 and segment 0 where strings were parsed into binary values.

```
select query, step, starttime, endtime, tasknum, rows
from stl_parse
where slice=1 and segment=0;
```

| query     | step | starttime           | endtime             | tasknum | rows   |
|-----------|------|---------------------|---------------------|---------|--------|
| 669       | 1    | 2013-08-12 22:35:13 | 2013-08-12 22:35:17 | 32      | 192497 |
| 696       | 1    | 2013-08-12 22:35:49 | 2013-08-12 22:35:49 | 32      | 0      |
| 525       | 1    | 2013-08-12 22:32:03 | 2013-08-12 22:32:03 | 13      | 49990  |
| 585       | 1    | 2013-08-12 22:33:18 | 2013-08-12 22:33:19 | 13      | 202    |
| 621       | 1    | 2013-08-12 22:34:03 | 2013-08-12 22:34:03 | 27      | 365    |
| 651       | 1    | 2013-08-12 22:34:47 | 2013-08-12 22:34:53 | 35      | 192497 |
| 590       | 1    | 2013-08-12 22:33:28 | 2013-08-12 22:33:28 | 19      | 0      |
| 599       | 1    | 2013-08-12 22:33:39 | 2013-08-12 22:33:39 | 31      | 11     |
| 675       | 1    | 2013-08-12 22:35:26 | 2013-08-12 22:35:27 | 38      | 3766   |
| 567       | 1    | 2013-08-12 22:32:47 | 2013-08-12 22:32:48 | 23      | 49990  |
| 630       | 1    | 2013-08-12 22:34:17 | 2013-08-12 22:34:17 | 36      | 0      |
| 572       | 1    | 2013-08-12 22:33:04 | 2013-08-12 22:33:04 | 29      | 0      |
| 645       | 1    | 2013-08-12 22:34:37 | 2013-08-12 22:34:38 | 29      | 8798   |
| 604       | 1    | 2013-08-12 22:33:47 | 2013-08-12 22:33:47 | 37      | 0      |
| (14 rows) |      |                     |                     |         |        |

## STL\_PLAN\_INFO

Use the STL\_PLAN\_INFO view to look at the EXPLAIN output for a query in terms of a set of rows. This is an alternative way to look at query plans.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                    |
|-------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                        |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                  |
| nodeid      | integer   | Plan node identifier, where a node maps to one or more steps in the execution of the query.                                                                                    |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                      |
| step        | integer   | Number that identifies the query step.                                                                                                                                         |
| locus       | integer   | Location where the step runs. 0 if on a compute node and 1 if on the leader node.                                                                                              |
| plannode    | integer   | Enumerated value of the plan node. See the following table for enums for plannode. (The PLANNODE column in <a href="#">STL_EXPLAIN (p. 1514)</a> contains the plan node text.) |

| Column name | Data type        | Description                                                           |
|-------------|------------------|-----------------------------------------------------------------------|
| startupcost | double precision | The estimated relative cost of returning the first row for this step. |
| totalcost   | double precision | The estimated relative cost of executing the step.                    |
| rows        | bigint           | The estimated number of rows that will be produced by the step.       |
| bytes       | bigint           | The estimated number of bytes that will be produced by the step.      |

## Sample queries

The following examples compare the query plans for a simple SELECT query returned by using the EXPLAIN command and by querying the STL\_PLAN\_INFO view.

```
explain select * from category;
QUERY PLAN

XN Seq Scan on category (cost=0.00..0.11 rows=11 width=49)
(1 row)

select * from category;
catid | catgroup | catname | catdesc
-----+-----+-----+
1 | Sports | MLB | Major League Baseball
3 | Sports | NFL | National Football League
5 | Sports | MLS | Major League Soccer
...
select * from stl_plan_info where query=256;

query | nodeid | segment | step | locus | plannode | startupcost | totalcost
| rows | bytes
-----+-----+-----+-----+-----+-----+-----+-----+
+-
256 | 1 | 0 | 1 | 0 | 104 | 0 | 0.11 | 11 | 539
256 | 1 | 0 | 0 | 0 | 104 | 0 | 0.11 | 11 | 539
(2 rows)
```

In this example, PLANNODE 104 refers to the sequential scan of the CATEGORY table.

```
select distinct eventname from event order by 1;

eventname

.38 Special
3 Doors Down
70s Soul Jam
A Bronx Tale
...
explain select distinct eventname from event order by 1;

QUERY PLAN

XN Merge (cost=1000000000136.38..1000000000137.82 rows=576 width=17)
Merge Key: eventname
-> XN Network (cost=1000000000136.38..1000000000137.82 rows=576
```

```

width=17)
Send to leader
-> XN Sort (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Sort Key: eventname
-> XN Unique (cost=0.00..109.98 rows=576 width=17)
-> XN Seq Scan on event (cost=0.00..87.98 rows=8798
width=17)
(8 rows)

select * from stl_plan_info where query=240 order by nodeid desc;

query | nodeid | segment | step | locus | plannode | startupcost | totalcost | rows | bytes
-----+-----+-----+-----+-----+-----+-----+-----+-----+
240 | 5 | 0 | 0 | 0 | 104 | 0 | 87.98 | 8798 | 149566
240 | 5 | 0 | 1 | 0 | 104 | 0 | 87.98 | 8798 | 149566
240 | 4 | 0 | 2 | 0 | 117 | 0 | 109.975 | 576 | 9792
240 | 4 | 0 | 3 | 0 | 117 | 0 | 109.975 | 576 | 9792
240 | 4 | 1 | 0 | 0 | 117 | 0 | 109.975 | 576 | 9792
240 | 4 | 1 | 1 | 0 | 117 | 0 | 109.975 | 576 | 9792
240 | 3 | 1 | 2 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 3 | 2 | 0 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 2 | 2 | 1 | 0 | 123 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 1 | 3 | 0 | 0 | 122 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
(10 rows)

```

## STL\_PROJECT

Contains rows for query steps that are used to evaluate expressions.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                             |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                       |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |

| Column name | Data type | Description                                |
|-------------|-----------|--------------------------------------------|
| rows        | bigint    | Total number of rows that were processed.  |
| checksum    | bigint    | This information is for internal use only. |

## Sample queries

The following example returns all rows for query steps that were used to evaluate expressions for slice 0 and segment 1.

```
select query, step, starttime, endtime, tasknum, rows
from stl_project
where slice=0 and segment=1;
```

| query     | step | starttime           | endtime             | tasknum | rows |
|-----------|------|---------------------|---------------------|---------|------|
| 86399     | 2    | 2013-08-29 22:01:21 | 2013-08-29 22:01:21 | 25      | -1   |
| 86399     | 3    | 2013-08-29 22:01:21 | 2013-08-29 22:01:21 | 25      | -1   |
| 719       | 1    | 2013-08-12 22:38:33 | 2013-08-12 22:38:33 | 7       | -1   |
| 86383     | 1    | 2013-08-29 21:58:35 | 2013-08-29 21:58:35 | 7       | -1   |
| 714       | 1    | 2013-08-12 22:38:17 | 2013-08-12 22:38:17 | 2       | -1   |
| 86375     | 1    | 2013-08-29 21:57:59 | 2013-08-29 21:57:59 | 2       | -1   |
| 86397     | 2    | 2013-08-29 22:01:20 | 2013-08-29 22:01:20 | 19      | -1   |
| 627       | 1    | 2013-08-12 22:34:13 | 2013-08-12 22:34:13 | 34      | -1   |
| 86326     | 2    | 2013-08-29 21:45:28 | 2013-08-29 21:45:28 | 34      | -1   |
| 86326     | 3    | 2013-08-29 21:45:28 | 2013-08-29 21:45:28 | 34      | -1   |
| 86325     | 2    | 2013-08-29 21:45:27 | 2013-08-29 21:45:27 | 28      | -1   |
| 86371     | 1    | 2013-08-29 21:57:42 | 2013-08-29 21:57:42 | 4       | -1   |
| 111100    | 2    | 2013-09-03 19:04:45 | 2013-09-03 19:04:45 | 12      | -1   |
| 704       | 2    | 2013-08-12 22:36:34 | 2013-08-12 22:36:34 | 37      | -1   |
| 649       | 2    | 2013-08-12 22:34:47 | 2013-08-12 22:34:47 | 38      | -1   |
| 649       | 3    | 2013-08-12 22:34:47 | 2013-08-12 22:34:47 | 38      | -1   |
| 632       | 2    | 2013-08-12 22:34:22 | 2013-08-12 22:34:22 | 13      | -1   |
| 705       | 2    | 2013-08-12 22:36:48 | 2013-08-12 22:36:49 | 13      | -1   |
| 705       | 3    | 2013-08-12 22:36:48 | 2013-08-12 22:36:49 | 13      | -1   |
| 3         | 1    | 2013-08-12 20:07:40 | 2013-08-12 20:07:40 | 3       | -1   |
| 86373     | 1    | 2013-08-29 21:57:58 | 2013-08-29 21:57:58 | 3       | -1   |
| 107976    | 1    | 2013-09-03 04:05:12 | 2013-09-03 04:05:12 | 3       | -1   |
| 86381     | 1    | 2013-08-29 21:58:35 | 2013-08-29 21:58:35 | 8       | -1   |
| 86396     | 1    | 2013-08-29 22:01:20 | 2013-08-29 22:01:20 | 15      | -1   |
| 711       | 1    | 2013-08-12 22:37:10 | 2013-08-12 22:37:10 | 20      | -1   |
| 86324     | 1    | 2013-08-29 21:45:27 | 2013-08-29 21:45:27 | 24      | -1   |
| (26 rows) |      |                     |                     |         |      |

## STL\_QUERY

Returns execution information about a database query.

### Note

The STL\_QUERY and STL\_QUERYTEXT views only contain information about queries, not other utility and DDL commands. For a listing and information on all statements run by Amazon Redshift, you can also query the STL\_DDLTEXT and STL.UtilityText views. For a complete listing of all statements run by Amazon Redshift, you can query the SVL\_STATEMENTTEXT view.

To manage disk space, the STL log views only retain approximately two to five days of log history, depending on log usage and available disk space. If you want to retain the log data, you will need to periodically copy it to other tables or unload it to Amazon S3.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name                | Data type       | Description                                                                                                                                                                                                                                                                                                                                                          |
|----------------------------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid                     | integer         | ID of the user who generated the entry.                                                                                                                                                                                                                                                                                                                              |
| query                      | integer         | Query ID. The query column can be used to join other system tables and views.                                                                                                                                                                                                                                                                                        |
| label                      | character(320)  | Either the name of the file used to run the query or a label defined with a SET QUERY_GROUP command. If the query is not file-based or the QUERY_GROUP parameter is not set, this field value is default.                                                                                                                                                            |
| xid                        | bigint          | Transaction ID.                                                                                                                                                                                                                                                                                                                                                      |
| pid                        | integer         | Process ID. Normally, all of the queries in a session are run in the same process, so this value usually remains constant if you run a series of queries in the same session. Following certain internal events, Amazon Redshift might restart an active session and assign a new PID. For more information, see <a href="#">STL_RESTARTED_SESSIONS (p. 1547)</a> .  |
| database                   | character(32)   | The name of the database the user was connected to when the query was issued.                                                                                                                                                                                                                                                                                        |
| querytxt                   | character(4000) | Actual query text for the query.                                                                                                                                                                                                                                                                                                                                     |
| starttime                  | timestamp       | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .                                                                                                                                                                                   |
| endtime                    | timestamp       | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .                                                                                                                                                                                  |
| aborted                    | integer         | If a query was stopped by the system or canceled by the user, this column contains <b>1</b> . If the query ran to completion (including returning results to the client), this column contains <b>0</b> . If a client disconnects before receiving the results, the query will be marked as canceled ( <b>1</b> ), even if it completed successfully in the backend. |
| insert_pristine            | integer         | Whether write queries are/were able to run while the current query is/was running. <b>1</b> = no write queries allowed. <b>0</b> = write queries allowed. This column is intended for use in debugging.                                                                                                                                                              |
| concurrency_scaling_status |                 | Indicates whether the query ran on the main cluster or on a concurrency scaling cluster. Possible values are as follows:<br><br><b>0</b> - Ran on the main cluster                                                                                                                                                                                                   |

| Column name | Data type | Description                              |
|-------------|-----------|------------------------------------------|
|             |           | 1 - Ran on a concurrency scaling cluster |
|             |           | Greater than 1 - Ran on the main cluster |

## Sample queries

The following query lists the five most recent queries.

```
select query, trim(querytxt) as sqlquery
from stl_query
order by query desc limit 5;

query | sqlquery
-----+-----
129 | select query, trim(querytxt) from stl_query order by query;
128 | select node from stv_disk_read_speeds;
127 | select system_status from stv_gui_status
126 | select * from systable_topology order by slice
125 | load global dict registry
(5 rows)
```

The following query returns the time elapsed in descending order for queries that ran on February 15, 2013.

```
select query, datediff(seconds, starttime, endtime),
trim(querytxt) as sqlquery
from stl_query
where starttime >= '2013-02-15 00:00' and endtime < '2013-02-16 00:00'
order by date_diff desc;

query | date_diff | sqlquery
-----+-----+
55 | 119 | padb_fetch_sample: select count(*) from category
121 | 9 | select * from svl_query_summary;
181 | 6 | select * from svl_query_summary where query in(179,178);
172 | 5 | select * from svl_query_summary where query=148;
...
(189 rows)
```

The following query shows the queue time and execution time for queries. Queries with `concurrency_scaling_status = 1` ran on a concurrency scaling cluster. All other queries ran on the main cluster.

```
SELECT w.service_class AS queue
, q.concurrency_scaling_status
, COUNT(*) AS queries
, SUM(q.aborted) AS aborted
, SUM(ROUND(total_queue_time::NUMERIC / 1000000,2)) AS queue_secs
, SUM(ROUND(total_exec_time::NUMERIC / 1000000,2)) AS exec_secs
FROM stl_query q
JOIN stl_wlm_query w
 USING (userid,query)
WHERE q.userid > 1
 AND service_class > 5
 AND q.starttime > '2019-03-01 16:38:00'
 AND q.endtime < '2019-03-01 17:40:00'
GROUP BY 1,2
```

```
ORDER BY 1,2;
```

## STL\_QUERY\_METRICS

Contains metrics information, such as the number of rows processed, CPU usage, input/output, and disk use, for queries that have completed running in user-defined query queues (service classes). To view metrics for active queries that are currently running, see the [STV\\_QUERY\\_METRICS \(p. 1592\)](#) system view.

Query metrics are sampled at one second intervals. As a result, different runs of the same query might return slightly different times. Also, query segments that run in less than one second might not be recorded.

STL\_QUERY\_METRICS tracks and aggregates metrics at the query, segment, and step level. For information about query segments and steps, see [Query planning and execution workflow \(p. 463\)](#). Many metrics (such as max\_rows, cpu\_time, and so on) are summed across node slices. For more information about node slices, see [Data warehouse system architecture \(p. 3\)](#).

To determine the level at which the row reports metrics, examine the segment and step\_type columns.

- If both segment and step\_type are -1, then the row reports metrics at the query level.
- If segment is not -1 and step\_type is -1, then the row reports metrics at the segment level.
- If both segment and step\_type are not -1, then the row reports metrics at the step level.

The [SVL\\_QUERY\\_METRICS \(p. 1642\)](#) view and the [SVL\\_QUERY\\_METRICS\\_SUMMARY \(p. 1643\)](#) view aggregate the data in this view and present the information in a more accessible form.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name   | Data type | Description                                                                                                                                                                                                                                                            |
|---------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid        | integer   | ID of the user that ran the query that generated the entry.                                                                                                                                                                                                            |
| service_class | integer   | ID for the service class. Query queues are defined in the WLM configuration. Metrics are reported only for user-defined queues.                                                                                                                                        |
| query         | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                                                                                                          |
| segment       | integer   | Segment number. A query consists of multiple segments, and each segment consists of one or more steps. Query segments can run in parallel. Each segment runs in a single process. If the segment value is -1, metrics segment values are rolled up to the query level. |
| step_type     | integer   | Type of step that ran. For a description of step types, see <a href="#">Step types (p. 1594)</a> .                                                                                                                                                                     |
| starttime     | timestamp | Time in UTC that the query started executing, with 6 digits of precision for fractional seconds. For example: 2009-06-12 11:29:19.131358.                                                                                                                              |
| slices        | integer   | Number of slices for the cluster.                                                                                                                                                                                                                                      |

| Column name         | Data type | Description                                                                                                                                                                                                                                                                                                                        |
|---------------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| max_rows            | bigint    | Maximum number of rows output for a step, aggregated across all slices.                                                                                                                                                                                                                                                            |
| rows                | bigint    | Number of rows processed by a step.                                                                                                                                                                                                                                                                                                |
| max_cpu_time        | bigint    | Maximum CPU time used, in microseconds. At the segment level, the maximum CPU time used by the segment across all slices. At the query level, the maximum CPU time used by any query segment.                                                                                                                                      |
| cpu_time            | bigint    | CPU time used, in microseconds. At the segment level, the total CPU time for the segment across all slices. At the query level, the sum of CPU time for the query across all slices and segments.                                                                                                                                  |
| max_blocks_read     | bigint    | Maximum number of 1 MB blocks read by the segment, aggregated across all slices. At the segment level, the maximum number of 1 MB blocks read for the segment across all slices. At the query level, the maximum number of 1 MB blocks read by any query segment.                                                                  |
| blocks_read         | bigint    | Number of 1 MB blocks read by the query or segment.                                                                                                                                                                                                                                                                                |
| max_run_time        | bigint    | The maximum elapsed time for a segment, in microseconds. At the segment level, the maximum run time for the segment across all slices. At the query level, the maximum run time for any query segment.                                                                                                                             |
| run_time            | bigint    | Total run time, summed across slices. Run time doesn't include wait time.<br><br>At the segment level, the run time for the segment, summed across all slices. At the query level, the run time for the query summed across all slices and segments. Because this value is a sum, run time is not related to query execution time. |
| max_blocks_to_disk  | bigint    | The maximum amount of disk space used to write intermediate results, in MB blocks. At the segment level, the maximum amount of disk space used by the segment across all slices. At the query level, the maximum amount of disk space used by any query segment.                                                                   |
| blocks_to_disk      | bigint    | The amount of disk space used by a query or segment to write intermediate results, in MB blocks.                                                                                                                                                                                                                                   |
| step                | integer   | Query step that ran.                                                                                                                                                                                                                                                                                                               |
| max_query_scan_size | bigint    | The maximum size of data scanned by a query, in MB. At the segment level, the maximum size of data scanned by the segment across all slices. At the query level, the maximum size of data scanned by any query segment.                                                                                                            |
| query_scan_size     | bigint    | The size of data scanned by a query, in MB.                                                                                                                                                                                                                                                                                        |
| query_priority      | integer   | The priority of the query. Possible values are -1, 0, 1, 2, 3, and 4, where -1 means that query priority isn't supported.                                                                                                                                                                                                          |
| query_queue_time    | bigint    | The amount of time in microseconds that the query was queued.                                                                                                                                                                                                                                                                      |

| Column name        | Data type     | Description                    |
|--------------------|---------------|--------------------------------|
| service_class_name | character(64) | The name of the service class. |

## Sample query

To find queries with high CPU time (more than 1,000 seconds), run the following query.

```
Select query, cpu_time / 1000000 as cpu_seconds
from stl_query_metrics where segment = -1 and cpu_time > 1000000000
order by cpu_time;

query | cpu_seconds
-----+-----
25775 | 9540
```

To find active queries with a nested loop join that returned more than one million rows, run the following query.

```
select query, rows
from stl_query_metrics
where step_type = 15 and rows > 1000000
order by rows;

query | rows
-----+-----
25775 | 2621562702
```

To find active queries that have run for more than 60 seconds and have used less than 10 seconds of CPU time, run the following query.

```
select query, run_time/1000000 as run_time_seconds
from stl_query_metrics
where segment = -1 and run_time > 60000000 and cpu_time < 10000000;

query | run_time_seconds
-----+-----
25775 | 114
```

## STL\_QUERYTEXT

Captures the query text for SQL commands.

Query the STL\_QUERYTEXT view to capture the SQL that was logged for the following statements:

- SELECT, SELECT INTO
- INSERT, UPDATE, DELETE
- COPY
- UNLOAD
- The queries generated by running VACUUM and ANALYZE
- CREATE TABLE AS (CTAS)

To query activity for these statements over a given time period, join the STL\_QUERYTEXT and STL\_QUERY views.

### Note

The STL\_QUERY and STL\_QUERYTEXT views only contain information about queries, not other utility and DDL commands. For a listing and information on all statements run by Amazon Redshift, you can also query the STL\_DDLTEXT and STL.UtilityText views. For a complete listing of all statements run by Amazon Redshift, you can query the SVL\_STATEMENTTEXT view.

See also [STL\\_DDLTEXT \(p. 1507\)](#), [STL.UtilityText \(p. 1567\)](#), and [SVL\\_STATEMENTTEXT \(p. 1661\)](#).

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer        | ID of the user who generated the entry.                                                                                                                                                                                                                                                                                                                                                                                                              |
| xid         | bigint         | Transaction ID.                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| pid         | integer        | Process ID. Normally, all of the queries in a session are run in the same process, so this value usually remains constant if you run a series of queries in the same session. Following certain internal events, Amazon Redshift might restart an active session and assign a new PID. For more information, see <a href="#">STL_RESTARTED_SESSIONS (p. 1547)</a> . You can use this column to join to the <a href="#">STL_ERROR (p. 1513)</a> view. |
| query       | integer        | Query ID. The query column can be used to join other system tables and views.                                                                                                                                                                                                                                                                                                                                                                        |
| sequence    | integer        | When a single statement contains more than 200 characters, additional rows are logged for that statement. Sequence 0 is the first row, 1 is the second, and so on.                                                                                                                                                                                                                                                                                   |
| text        | character(200) | SQL text, in 200-character increments. This field might contain special characters such as backslash (\\\) and newline (\n).                                                                                                                                                                                                                                                                                                                         |

### Sample queries

You can use the PG\_BACKEND\_PID() function to retrieve information for the current session. For example, the following query returns the query ID and a portion of the query text for queries completed in the current session.

```
select query, substring(text,1,60)
from stl_querytext
where pid = pg_backend_pid()
order by query desc;

query | substring
-----+-----
28262 | select query, substring(text,1,80) from stl_querytext where
28252 | select query, substring(path,0,80) as path from stl_unload_1
28248 | copy category from 's3://dw-ticket/manifest/category/1030_ma
28247 | Count rows in target table
28245 | unload ('select * from category') to 's3://dw-ticket/manifes
28240 | select query, substring(text,1,40) from stl_querytext where
```

(6 rows)

## Reconstructing stored SQL

To reconstruct the SQL stored in the text column of STL\_QUERYTEXT, run a SELECT statement to create SQL from 1 or more parts in the text column. Before running the reconstructed SQL, replace any (\n) special characters with a new line. The result of the following SELECT statement is rows of reconstructed SQL in the query\_statement field.

```
SELECT query, LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END) WITHIN GROUP (ORDER BY sequence) as query_statement, COUNT(*) as row_count FROM stl_querytext GROUP BY query ORDER BY query desc;
```

For example, the following query selects 3 columns. The query itself is longer than 200 characters and is stored in parts in STL\_QUERYTEXT.

```
select
1 AS a0123456789012345678901234567890123456789012345678901234567890,
2 AS b0123456789012345678901234567890123456789012345678901234567890,
3 AS b0123456789012345678901234567890123456789012345678901234
FROM stl_querytext;
```

In this example, the query is stored in 2 parts (rows) in the text column of STL\_QUERYTEXT.

```
select query, sequence, text
from stl_querytext where query=pg_last_query_id() order by query desc, sequence limit 10;
```

| query | sequence | text                                                                                                                                                               |
|-------|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 45    | 0        | select\n1 AS a01234567890123456789012345678901234567890,\n2 AS b01234567890123456789012345678901234567890,\n3 AS b012345678901234567890123456789012345678901234 45 |
|       | 1        | \nFROM stl_querytext;                                                                                                                                              |

To reconstruct the SQL stored in STL\_QUERYTEXT, run the following SQL.

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '') within group (order by sequence) AS text
from stl_querytext where query=pg_last_query_id();
```

To use the resulting reconstructed SQL in your client, replace any (\n) special characters with a new line.

| text                                                                                                                                                                                                     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| select\n1 AS a012345678901234567890123456789012345678901234567890,\n2 AS b012345678901234567890123456789012345678901234567890,\n3 AS b012345678901234567890123456789012345678901234\nFROM stl_querytext; |

## STL\_REPLACEMENTS

Displays a log that records when invalid UTF-8 characters were replaced by the [COPY \(p. 677\)](#) command with the ACCEPTINVCHARS option. A log entry is added to STL\_REPLACEMENTS for each of the first 100 rows on each node slice that required at least one replacement.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type       | Description                                                                                                                                                                          |
|-------------|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer         | ID of the user who generated the entry.                                                                                                                                              |
| query       | integer         | Query ID. The query column can be used to join other system tables and views.                                                                                                        |
| slice       | integer         | Node slice number where the replacement occurred.                                                                                                                                    |
| tbl         | integer         | Table ID.                                                                                                                                                                            |
| starttime   | timestamp       | Start time in UTC for the COPY command.                                                                                                                                              |
| session     | integer         | Session ID for the session performing the COPY command.                                                                                                                              |
| filename    | character(256)  | Complete path to the input file for the COPY command.                                                                                                                                |
| line_number | bigint          | Line number in the input data file that contained an invalid UTF-8 character. A -1 indicates that the line number is not available, such as, when copying from a columnar data file. |
| colname     | character(127)  | First field that contained an invalid UTF-8 character.                                                                                                                               |
| raw_line    | character(1024) | Raw load data that contained an invalid UTF-8 character.                                                                                                                             |

### Sample queries

The following example returns replacements for the most recent COPY operation.

```
select query, session, filename, line_number, colname
from stl_replacements
where query = pg_last_copy_id();

query | session | filename | line_number | colname
-----+-----+-----+-----+-----+
 96 | 6314 | s3://mybucket/allusers_pipe.txt | 251 | city
 96 | 6314 | s3://mybucket/allusers_pipe.txt | 317 | city
 96 | 6314 | s3://mybucket/allusers_pipe.txt | 569 | city
 96 | 6314 | s3://mybucket/allusers_pipe.txt | 623 | city
 96 | 6314 | s3://mybucket/allusers_pipe.txt | 694 | city
...
```

## STL\_RESTARTED\_SESSIONS

To maintain continuous availability following certain internal events, Amazon Redshift might restart an active session with a new process ID (PID). When Amazon Redshift restarts a session, STL\_RESTARTED\_SESSIONS records the new PID and the old PID.

For more information, see the examples following in this section.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name  | Data type       | Description                                       |
|--------------|-----------------|---------------------------------------------------|
| currenttime  | timestamp       | Time of the event.                                |
| dbname       | character(50)   | Name of the database associated with the session. |
| newpid       | integer         | Process ID for the restarted session.             |
| oldpid       | integer         | Process ID for the original session.              |
| username     | character(50)   | Name of the user associated with the session.     |
| remotehost   | character(32)   | Name or IP address of the remote host.            |
| remoteport   | character(32)   | Port number of the remote host.                   |
| parkedtime   | timestamp       | This information is for internal use only.        |
| session_vars | character(2000) | This information is for internal use only.        |

## Sample queries

The following example joins STL\_RESTARTED\_SESSIONS with STL\_SESSIONS to show user names for sessions that have been restarted.

```
select process, stl_restarted_sessions.newpid, user_name
from stl_sessions
inner join stl_restarted_sessions on stl_sessions.process = stl_restarted_sessions.oldpid
order by process;

...
```

## STL\_RETURN

Contains details for *return* steps in queries. A return step returns the results of queries completed on the compute nodes to the leader node. The leader node then merges the data and returns the results to the requesting client. For queries completed on the leader node, a return step returns results to the client.

A query consists of multiple segments, and each segment consists of one or more steps. For more information, see [Query processing \(p. 463\)](#).

STL\_RETURN is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type | Description                             |
|-------------|-----------|-----------------------------------------|
| userid      | integer   | ID of the user who generated the entry. |

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                       |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows        | bigint    | Total number of rows that were processed.                                                                                                                                           |
| bytes       | bigint    | Size, in bytes, of all the output rows for the step.                                                                                                                                |
| packets     | integer   | Total number of packets sent over the network.                                                                                                                                      |
| checksum    | bigint    | This information is for internal use only.                                                                                                                                          |

## Sample queries

The following query shows which steps in the most recent query were performed on each slice.

```
SELECT query, slice, segment, step, endtime, rows, packets
from stl_return where query = pg_last_query_id();

query | slice | segment | step | endtime | rows | packets
-----+-----+-----+-----+-----+-----+-----+
 4 | 2 | 3 | 2 | 2013-12-27 01:43:21.469043 | 3 | 0
 4 | 3 | 3 | 2 | 2013-12-27 01:43:21.473321 | 0 | 0
 4 | 0 | 3 | 2 | 2013-12-27 01:43:21.469118 | 2 | 0
 4 | 1 | 3 | 2 | 2013-12-27 01:43:21.474196 | 0 | 0
 4 | 4 | 3 | 2 | 2013-12-27 01:43:21.47704 | 2 | 0
 4 | 5 | 3 | 2 | 2013-12-27 01:43:21.478593 | 0 | 0
 4 | 12811 | 4 | 1 | 2013-12-27 01:43:21.480755 | 0 | 0
(7 rows)
```

## STL\_S3CLIENT

Records transfer time and other performance metrics.

Use the STL\_S3CLIENT table to find the time spent transferring data from Amazon S3 as part of a COPY command.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name         | Data type      | Description                                                                                                                                                 |
|---------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid              | integer        | ID of the user who generated the entry.                                                                                                                     |
| query               | integer        | Query ID. The query column can be used to join other system tables and views.                                                                               |
| slice               | integer        | Number that identifies the slice where the query was running.                                                                                               |
| recordtime          | timestamp      | Time the record is logged.                                                                                                                                  |
| pid                 | integer        | Process ID. All of the queries in a session are run in the same process, so this value remains constant if you run a series of queries in the same session. |
| http_method         | character(64)  | HTTP method name corresponding to the Amazon S3 request.                                                                                                    |
| bucket              | character(64)  | S3 bucket name.                                                                                                                                             |
| key                 | character(256) | Key corresponding to the Amazon S3 object.                                                                                                                  |
| transfer_size       | bigint         | Number of bytes transferred.                                                                                                                                |
| data_size           | bigint         | Number of bytes of data. This value is the same as transfer_size for uncompressed data. If compression was used, this is the size of the uncompressed data. |
| start_time          | bigint         | Time when the transfer began (in microseconds since January 1, 2000).                                                                                       |
| end_time            | bigint         | Time when the transfer ended (in microseconds since January 1, 2000).                                                                                       |
| transfer_time       | bigint         | Time taken by the transfer (in microseconds).                                                                                                               |
| compression_time    | bigint         | Portion of the transfer time that was spent uncompressing data (in microseconds).                                                                           |
| connect_time        | bigint         | Time from the start until the connect to the remote server was completed (in microseconds).                                                                 |
| app_connect_time    | bigint         | Time from the start until the SSL connect/handshake with the remote host was completed (in microseconds).                                                   |
| retries             | bigint         | Number of times the transfer was retried.                                                                                                                   |
| request_id          | char(32)       | Request ID from Amazon S3 HTTP response header                                                                                                              |
| extended_request_id | char(128)      | Extended request ID from Amazon S3 HTTP header response (x-amz-id-2).                                                                                       |
| ip_address          | char(64)       | IP address of the server (ip V4 or V6).                                                                                                                     |
| is_partial          | integer        | Value that if true (1) indicates the input file is split into ranges during a COPY operation. If this value is false (0), the input file isn't split.       |

| Column name  | Data type | Description                                                                                                                                                   |
|--------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| start_offset | bigint    | Value that, if the input file is split during a COPY operation, indicates the offset value of the split (in bytes). If the file isn't split, this value is 0. |

## Sample query

The following query returns the time taken to load files using a COPY command.

```
select slice, key, transfer_time
from stl_s3client
where query = pg_last_copy_id();
```

### Result

| slice | key                       | transfer_time |
|-------|---------------------------|---------------|
| 0     | listing10M0003_part_00    | 16626716      |
| 1     | listing10M0001_part_00    | 12894494      |
| 2     | listing10M0002_part_00    | 14320978      |
| 3     | listing10M0000_part_00    | 11293439      |
| 3371  | prefix=listing10M;marker= | 99395         |

The following query converts the start\_time and end\_time to a timestamp.

```
select userid,query,slice,pid,recordtime,start_time,end_time,
'2000-01-01':timestamp + (start_time/1000000.0)* interval '1 second' as start_ts,
'2000-01-01':timestamp + (end_time/1000000.0)* interval '1 second' as end_ts
from stl_s3client where query > -1 limit 5;
```

| userid          | query | slice | pid   | recordtime                 | start_time                 | end_time        |
|-----------------|-------|-------|-------|----------------------------|----------------------------|-----------------|
| 0               | 0     | 0     | 23449 | 2019-07-14 16:27:17.207839 | 616436837154256            | 616436837207838 |
| 616436837207838 |       |       |       | 2019-07-14 16:27:17.154256 | 2019-07-14 16:27:17.207838 |                 |
| 0               | 0     | 0     | 23449 | 2019-07-14 16:27:17.252521 | 616436837208208            | 616436837252520 |
| 616436837252520 |       |       |       | 2019-07-14 16:27:17.208208 | 2019-07-14 16:27:17.25252  |                 |
| 0               | 0     | 0     | 23449 | 2019-07-14 16:27:17.284376 | 616436837208460            | 616436837284374 |
| 616436837284374 |       |       |       | 2019-07-14 16:27:17.20846  | 2019-07-14 16:27:17.284374 |                 |
| 0               | 0     | 0     | 23449 | 2019-07-14 16:27:17.285307 | 616436837208980            | 616436837285306 |
| 616436837285306 |       |       |       | 2019-07-14 16:27:17.20898  | 2019-07-14 16:27:17.285306 |                 |
| 0               | 0     | 0     | 23449 | 2019-07-14 16:27:17.353853 | 616436837302216            | 616436837353851 |
| 616436837353851 |       |       |       | 2019-07-14 16:27:17.302216 | 2019-07-14 16:27:17.353851 |                 |

## STL\_S3CLIENT\_ERROR

Records errors encountered by a slice while loading a file from Amazon S3.

Use the STL\_S3CLIENT\_ERROR to find details for errors encountered while transferring data from Amazon S3 as part of a COPY command.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name  | Data type       | Description                                                                                                                                                   |
|--------------|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid       | integer         | ID of the user who generated the entry.                                                                                                                       |
| query        | integer         | Query ID. The query column can be used to join other system tables and views.                                                                                 |
| sliceid      | integer         | Number that identifies the slice where the query was running.                                                                                                 |
| recordtime   | timestamp       | Time the record is logged.                                                                                                                                    |
| pid          | integer         | Process ID. All of the queries in a session are run in the same process, so this value remains constant if you run a series of queries in the same session.   |
| http_method  | character(64)   | HTTP method name corresponding to the Amazon S3 request.                                                                                                      |
| bucket       | character(64)   | Amazon S3 bucket name.                                                                                                                                        |
| key          | character(256)  | Key corresponding to the Amazon S3 object.                                                                                                                    |
| error        | character(1024) | Error message.                                                                                                                                                |
| is_partial   | integer         | Value that, if true (1), indicates the input file is split into ranges during a COPY operation. If this value is false (0), the input file isn't split.       |
| start_offset | bigint          | Value that, if the input file is split during a COPY operation, indicates the offset value of the split (in bytes). If the file isn't split, this value is 0. |

## Usage notes

If you see multiple errors with "Connection timed out", you might have a networking issue. If you're using Enhanced VPC Routing, verify that you have a valid network path between your cluster's VPC and your data resources. For more information, see [Amazon Redshift Enhanced VPC Routing](#).

## Sample query

The following query returns the errors from COPY commands completed during the current session.

```
select query, sliceid, substring(key from 1 for 20) as file,
substring(error from 1 for 35) as error
from stl_s3client_error
where pid = pg_backend_pid()
order by query desc;
```

### Result

| query  | sliceid | file               | error                               |
|--------|---------|--------------------|-------------------------------------|
| 362228 | 12      | part.tbl.25.159.gz | transfer closed with 1947655 bytes  |
| 362228 | 24      | part.tbl.15.577.gz | transfer closed with 1881910 bytes  |
| 362228 | 7       | part.tbl.22.600.gz | transfer closed with 700143 bytes r |
| 362228 | 22      | part.tbl.3.34.gz   | transfer closed with 2334528 bytes  |

|           |    |                                                          |
|-----------|----|----------------------------------------------------------|
| 362228    | 11 | part.tbl.30.274.gz   transfer closed with 699031 bytes r |
| 362228    | 30 | part.tbl.5.509.gz   Unknown SSL protocol error in conne  |
| 361999    | 10 | part.tbl.23.305.gz   transfer closed with 698959 bytes r |
| 361999    | 19 | part.tbl.26.582.gz   transfer closed with 1881458 bytes  |
| 361999    | 4  | part.tbl.15.629.gz   transfer closed with 2275907 bytes  |
| 361999    | 20 | part.tbl.6.456.gz   transfer closed with 692162 bytes r  |
| (10 rows) |    |                                                          |

## STL\_SAVE

Contains details for *save* steps in queries. A save step saves the input stream to a transient table. A transient table is a temporary table that stores intermediate results during query execution.

A query consists of multiple segments, and each segment consists of one or more steps. For more information, see [Query processing \(p. 463\)](#).

STL\_SAVE is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name  | Data type    | Description                                                                                                                                                                         |
|--------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid       | integer      | ID of the user who generated the entry.                                                                                                                                             |
| query        | integer      | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice        | integer      | Number that identifies the slice where the query was running.                                                                                                                       |
| segment      | integer      | Number that identifies the query segment.                                                                                                                                           |
| step         | integer      | Query step that ran.                                                                                                                                                                |
| starttime    | timestamp    | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime      | timestamp    | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum      | integer      | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows         | bigint       | Total number of rows that were processed.                                                                                                                                           |
| bytes        | bigint       | Size, in bytes, of all the output rows for the step.                                                                                                                                |
| tbl          | integer      | ID of the materialized transient table.                                                                                                                                             |
| is_diskbased | character(1) | Whether this step of the query was performed as a disk-based operation: true (t) or false (f).                                                                                      |
| workmem      | bigint       | Number of bytes of working memory assigned to the step.                                                                                                                             |

### Sample queries

The following query shows which save steps in the most recent query were performed on each slice.

```
select query, slice, segment, step, tasknum, rows, tbl
from stl_save where query = pg_last_query_id();

query | slice | segment | step | tasknum | rows | tbl
-----+-----+-----+-----+-----+-----+
52236 | 3 | 0 | 2 | 21 | 0 | 239
52236 | 2 | 0 | 2 | 20 | 0 | 239
52236 | 2 | 2 | 2 | 20 | 0 | 239
52236 | 3 | 2 | 2 | 21 | 0 | 239
52236 | 1 | 0 | 2 | 21 | 0 | 239
52236 | 0 | 0 | 2 | 20 | 0 | 239
52236 | 0 | 2 | 2 | 20 | 0 | 239
52236 | 1 | 2 | 2 | 21 | 0 | 239
(8 rows)
```

## STL\_SCAN

Analyzes table scan steps for queries. The step number for rows in this table is always 0 because a scan is the first step in a segment.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                             |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                       |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows        | bigint    | Total number of rows that were processed.                                                                                                                                           |
| bytes       | bigint    | Size, in bytes, of all the output rows for the step.                                                                                                                                |
| fetches     | bigint    | This information is for internal use only.                                                                                                                                          |
| type        | integer   | ID of the scan type. For a list of valid values, see the following table.                                                                                                           |
| tbl         | integer   | Table ID.                                                                                                                                                                           |

| Column name                  | Data type      | Description                                                                                                                                                                                                                                                       |
|------------------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| is_rrscan                    | character(1)   | If true (t), indicates that range-restricted scan was used on the step.                                                                                                                                                                                           |
| is_delayed_scan              | character(1)   | This information is for internal use only.                                                                                                                                                                                                                        |
| rows_pre_filter              | bigint         | For scans of permanent tables, the total number of rows emitted before filtering rows marked for deletion (ghost rows) and before applying user-defined query filters.                                                                                            |
| rows_pre_user_filter         | bigint         | For scans of permanent tables, the number of rows processed after filtering rows marked for deletion (ghost rows) but before applying user-defined query filters.                                                                                                 |
| perm_table_name              | character(136) | For scans of permanent tables, the name of the table scanned.                                                                                                                                                                                                     |
| is_rlf_scan                  | character(1)   | If true (t), indicates that row-level filtering was used on the step.                                                                                                                                                                                             |
| is_rlf_scan_reason           | integer        | This information is for internal use only.                                                                                                                                                                                                                        |
| num_em_blocks                | integer        | This information is for internal use only.                                                                                                                                                                                                                        |
| checksum                     | bigint         | This information is for internal use only.                                                                                                                                                                                                                        |
| runtime_filtering            | character(1)   | If true (t), indicates that runtime filters are applied.                                                                                                                                                                                                          |
| scan_region                  | integer        | This information is for internal use only.                                                                                                                                                                                                                        |
| num_sortkey_partitions       | integer        | This information is for internal use only.                                                                                                                                                                                                                        |
| row_fetcher_start            | integer        | This information is for internal use only.                                                                                                                                                                                                                        |
| consumed_scans               | bigint         | This information is for internal use only.                                                                                                                                                                                                                        |
| work_stealing_blocks         | bigint         | This information is for internal use only.                                                                                                                                                                                                                        |
| is_vectorized_scanner        | character(1)   | This information is for internal use only.                                                                                                                                                                                                                        |
| is_vectorized_scanner_reason | integer        | This information is for internal use only.                                                                                                                                                                                                                        |
| row_fetcher_rebind           | bigint         | This information is for internal use only.                                                                                                                                                                                                                        |
| topology_signature           | bigint         | This information is for internal use only.                                                                                                                                                                                                                        |
| use_tpm_partition            | character(1)   | This information is for internal use only.                                                                                                                                                                                                                        |
| is_rrscan_expr               | character(1)   | This information is for internal use only.                                                                                                                                                                                                                        |
| scanned_megabytes            | character(1)   | This information is for internal use only. This information shows whether the given scan step has scanned a large value. A large value will be stored in multiple blocks. Block size is 1 MB by default, a large value is greater than 1 MB in a default setting. |

## Scan types

| Type ID | Description                                                                |
|---------|----------------------------------------------------------------------------|
| 1       | Data from the network.                                                     |
| 2       | Permanent user tables in compressed shared memory.                         |
| 3       | Transient row-wise tables.                                                 |
| 21      | Load files from Amazon S3.                                                 |
| 22      | Load tables from Amazon DynamoDB.                                          |
| 23      | Load data from a remote SSH connection.                                    |
| 24      | Load data from remote cluster (sorted region). This is used for resizing.  |
| 25      | Load data from remote cluster(unsorted region). This is used for resizing. |
| 28      | Read data from a time series view with UNION ALL on multiple tables.       |
| 29      | Read data from Amazon S3 external tables.                                  |
| 30      | Read partition information of an Amazon S3 external table.                 |
| 33      | Read data from a remote Postgres table.                                    |
| 36      | Read data from a remote MySQL table.                                       |
| 37      | Read data from a remote Kinesis stream.                                    |

## Usage notes

Ideally `rows` should be relatively close to `rows_pre_filter`. A large difference between `rows` and `rows_pre_filter` is an indication that the execution engine is scanning rows that are later discarded, which is inefficient. The difference between `rows_pre_filter` and `rows_pre_user_filter` is the number of ghost rows in the scan. Run a VACUUM to remove rows marked for deletion. The difference between `rows` and `rows_pre_user_filter` is the number of rows filtered by the query. If a lot of rows are discarded by the user filter, review your choice of sort column or, if this is due to a large unsorted region, run a vacuum.

## Sample queries

The following example shows that `rows_pre_filter` is larger than `rows_pre_user_filter` because the table has deleted rows that have not been vacuumed (ghost rows).

```
SELECT query, slice, segment, step, rows, rows_pre_filter, rows_pre_user_filter
from stl_scan where query = pg_last_query_id();

query | slice | segment | step | rows | rows_pre_filter | rows_pre_user_filter
-----+-----+-----+-----+-----+-----+-----+
42915 | 0 | 0 | 0 | 43159 | 86318 | 43159
42915 | 0 | 1 | 0 | 1 | 0 | 0
42915 | 1 | 0 | 0 | 43091 | 86182 | 43091
42915 | 1 | 1 | 0 | 1 | 0 | 0
42915 | 2 | 0 | 0 | 42778 | 85556 | 42778
42915 | 2 | 1 | 0 | 1 | 0 | 0
42915 | 3 | 0 | 0 | 43428 | 86856 | 43428
42915 | 3 | 1 | 0 | 1 | 0 | 0
```

|                                   |
|-----------------------------------|
| 42915   10000   2   0   4   0   0 |
| (9 rows)                          |

## STL\_SCHEMA\_QUOTA\_VIOLATIONS

Records the occurrence, timestamp, XID, and other useful information when a schema quota is exceeded.

Superusers can see all the records. Schema owners can only see records related to the schemas they own.

### Table columns

| Column name    | Data type                   | Description                                                                                 |
|----------------|-----------------------------|---------------------------------------------------------------------------------------------|
| ownerid        | integer                     | The ID of the schema owner.                                                                 |
| xid            | bigint                      | The transaction ID associated with the statement.                                           |
| pid            | integer                     | The process ID associated with the statement.                                               |
| userid         | integer                     | The ID of the user who generated the entry.                                                 |
| schema_id      | integer                     | The namespace or schema ID.                                                                 |
| schema_name    | character (128)             | The namespace or schema name.                                                               |
| quota          | integer                     | The amount of disk space (in MB) that the schema can use.                                   |
| disk_usage     | integer                     | The disk space (in MB) that is currently used by the schema.                                |
| disk_usage_pct | double precision            | The disk space percentage that is currently used by the schema out of the configured quota. |
| timestamp      | timestamp without time zone | The time when the violation occurred.                                                       |

### Sample queries

The following query shows the result of quota violation:

```
SELECT userid, TRIM(SCHEMA_NAME) "schema_name", quota, disk_usage, disk_usage_pct,
 timestamp
 FROM stl_schema_quotaViolations
 WHERE SCHEMA_NAME = 'sales_schema'
 ORDER BY timestamp DESC;
```

This query returns the following sample output for the specified schema:

|                                                                        |
|------------------------------------------------------------------------|
| userid   schema_name   quota   disk_usage   disk_usage_pct   timestamp |
| -----+-----+-----+-----+-----+-----                                    |
| 104   sales_schema   2048   2798   136.62   2020-04-20 20:09:25.494723 |
| (1 row)                                                                |

## STL\_SESSIONS

Returns information about user session history.

STL\_SESSIONS differs from STV\_SESSIONS in that STL\_SESSIONS contains session history, where STV\_SESSIONS contains the current active sessions.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type     | Description                                                                                                                |
|-------------|---------------|----------------------------------------------------------------------------------------------------------------------------|
| userid      | integer       | ID of the user who generated the entry.                                                                                    |
| starttime   | timestamp     | Time in UTC that the session started.                                                                                      |
| endtime     | timestamp     | Time in UTC that the session ended.                                                                                        |
| process     | integer       | Process ID for the session.                                                                                                |
| user_name   | character(50) | User name associated with the session.                                                                                     |
| db_name     | character(50) | Name of the database associated with the session.                                                                          |
| timeout_sec | int           | The maximum time in seconds that a session remains inactive or idle before timing out. 0 indicates that no timeout is set. |
| timed_out   | int           | A value that indicates if a session has timed out: 1 if it has timed out, 0 otherwise.                                     |

## Sample queries

To view session history for the TICKIT database, type the following query:

```
select starttime, process, user_name, timeout_sec, timed_out
from stl_sessions
where db_name='tickit' order by starttime;
```

This query returns the following sample output:

| starttime                  | process | user_name | timeout_sec | timed_out |
|----------------------------|---------|-----------|-------------|-----------|
| 2008-09-15 09:54:06.746705 | 32358   | dwuser    | 120         | 1         |
| 2008-09-15 09:56:34.30275  | 32744   | dwuser    | 60          | 1         |
| 2008-09-15 11:20:34.694837 | 14906   | dwuser    | 0           | 0         |
| 2008-09-15 11:22:16.749818 | 15148   | dwuser    | 0           | 0         |
| 2008-09-15 14:32:44.66112  | 14031   | dwuser    | 0           | 0         |
| 2008-09-15 14:56:30.22161  | 18380   | dwuser    | 0           | 0         |
| 2008-09-15 15:28:32.509354 | 24344   | dwuser    | 0           | 0         |
| 2008-09-15 16:01:00.557326 | 30153   | dwuser    | 120         | 1         |
| 2008-09-15 17:28:21.419858 | 12805   | dwuser    | 0           | 0         |
| 2008-09-15 20:58:37.601937 | 14951   | dwuser    | 60          | 1         |
| 2008-09-16 11:12:30.960564 | 27437   | dwuser    | 60          | 1         |
| 2008-09-16 14:11:37.639092 | 23790   | dwuser    | 3600        | 1         |
| 2008-09-16 15:13:46.02195  | 1355    | dwuser    | 120         | 1         |
| 2008-09-16 15:22:36.515106 | 2878    | dwuser    | 120         | 1         |
| 2008-09-16 15:44:39.194579 | 6470    | dwuser    | 120         | 1         |
| 2008-09-16 16:50:27.02138  | 17254   | dwuser    | 120         | 1         |

|                                            |      |   |
|--------------------------------------------|------|---|
| 2008-09-17 12:05:02.157208   8439   dwuser | 3600 | 0 |
| (17 rows)                                  |      |   |

## STL\_SORT

Displays sort execution steps for queries, such as steps that use ORDER BY processing.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name  | Data type    | Description                                                                                                                                                                         |
|--------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid       | integer      | ID of the user who generated the entry.                                                                                                                                             |
| query        | integer      | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice        | integer      | Number that identifies the slice where the query was running.                                                                                                                       |
| segment      | integer      | Number that identifies the query segment.                                                                                                                                           |
| step         | integer      | Query step that ran.                                                                                                                                                                |
| starttime    | timestamp    | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime      | timestamp    | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum      | integer      | Number of the query task process that was assigned to run the step.                                                                                                                 |
| rows         | bigint       | Total number of rows that were processed.                                                                                                                                           |
| bytes        | bigint       | Size, in bytes, of all the output rows for the step.                                                                                                                                |
| tbl          | integer      | Table ID.                                                                                                                                                                           |
| is_diskbased | character(1) | If true (t), the query was performed as a disk-based operation. If false (f), the query was performed in memory.                                                                    |
| workmem      | bigint       | Total number of bytes in working memory that were assigned to the step.                                                                                                             |
| checksum     | bigint       | This information is for internal use only.                                                                                                                                          |

### Sample queries

The following example returns sort results for slice 0 and segment 1.

```
select query, bytes, tbl, is_diskbased, workmem
from stl_sort
where slice=0 and segment=1;
```

| query     | bytes   | tbl | is_diskbased | workmem   |
|-----------|---------|-----|--------------|-----------|
| 567       | 3126968 | 241 | f            | 383385600 |
| 604       | 5292    | 242 | f            | 383385600 |
| 675       | 104776  | 251 | f            | 383385600 |
| 525       | 3126968 | 251 | f            | 383385600 |
| 585       | 5068    | 241 | f            | 383385600 |
| 630       | 204808  | 266 | f            | 383385600 |
| 704       | 0       | 242 | f            | 0         |
| 669       | 4606416 | 241 | f            | 383385600 |
| 696       | 104776  | 241 | f            | 383385600 |
| 651       | 4606416 | 254 | f            | 383385600 |
| 632       | 0       | 256 | f            | 0         |
| 599       | 396     | 241 | f            | 383385600 |
| 86397     | 0       | 242 | f            | 0         |
| 621       | 5292    | 241 | f            | 383385600 |
| 86325     | 0       | 242 | f            | 0         |
| 572       | 5068    | 242 | f            | 383385600 |
| 645       | 204808  | 241 | f            | 383385600 |
| 590       | 396     | 242 | f            | 383385600 |
| (18 rows) |         |     |              |           |

## STL\_SSCLIENT\_ERROR

Records all errors seen by the SSH client.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name  | Data type       | Description                                                                   |
|--------------|-----------------|-------------------------------------------------------------------------------|
| userid       | integer         | ID of the user who generated the entry.                                       |
| query        | integer         | Query ID. The query column can be used to join other system tables and views. |
| slice        | integer         | Number that identifies the slice where the query was running.                 |
| recordtime   | timestamp       | Time that the error was logged.                                               |
| pid          | integer         | Process that logged the error.                                                |
| ssh_username | character(1024) | The SSH user name.                                                            |
| endpoint     | character(1024) | The SSH endpoint.                                                             |
| command      | character(4096) | The complete SSH command.                                                     |
| error        | character(1024) | The error message.                                                            |

## STL\_STREAM\_SEGS

Lists the relationship between streams and concurrent segments.

Streams in this context are Amazon Redshift streams. This system view doesn't pertain to [Streaming ingestion \(p. 208\)](#).

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type | Description                                                                   |
|-------------|-----------|-------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                       |
| query       | integer   | Query ID. The query column can be used to join other system tables and views. |
| stream      | integer   | The set of concurrent segments of a query.                                    |
| segment     | integer   | Number that identifies the query segment.                                     |

## Sample queries

To view the relationship between streams and concurrent segments for the most recent query, type the following query:

```
select *
from stl_stream_segs
where query = pg_last_query_id();

query | stream | segment
-----+-----+-----
 10 | 1 | 2
 10 | 0 | 0
 10 | 2 | 4
 10 | 1 | 3
 10 | 0 | 1
(5 rows)
```

## STL\_TR\_CONFLICT

Displays information to identify and resolve transaction conflicts with database tables.

A transaction conflict occurs when two or more users are querying and modifying data rows from tables such that their transactions cannot be serialized. The transaction that runs a statement that would break serializability is stopped and rolled back. Every time a transaction conflict occurs, Amazon Redshift writes a data row to the STL\_TR\_CONFLICT system table containing details about the canceled transaction. For more information, see [Serializable isolation \(p. 119\)](#).

This view is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type | Description                                                   |
|-------------|-----------|---------------------------------------------------------------|
| xact_id     | bigint    | Transaction ID for the rolled back transaction.               |
| process_id  | bigint    | Process associated with the transaction that was rolled back. |

| Column name   | Data type | Description                                         |
|---------------|-----------|-----------------------------------------------------|
| xact_start_ts | timestamp | Timestamp for the transaction start.                |
| abort_time    | timestamp | Time that the transaction was stopped.              |
| table_id      | bigint    | Table ID for the table where the conflict occurred. |

## Sample query

To return information about conflicts that involved a particular table, run a query that specifies the table ID:

```
select * from stl_tr_conflict where table_id=100234
order by xact_start_ts;

xact_id|process_| xact_start_ts | abort_time |table_
 |id | | |id
-----+-----+-----+-----+-----+-----+
 1876 | 8551 | 2010-03-30 09:19:15.852326 | 2010-03-30 09:20:17.582499 | 100234
 1928 | 15034 | 2010-03-30 13:20:00.636045 | 2010-03-30 13:20:47.766817 | 100234
 1991 | 23753 | 2010-04-01 13:05:01.220059 | 2010-04-01 13:06:06.94098 | 100234
 2002 | 23679 | 2010-04-01 13:17:05.173473 | 2010-04-01 13:18:27.898655 | 100234
(4 rows)
```

You can get the table ID from the DETAIL section of the error message for serializability violations (error 1023).

## STL\_UNDONE

Displays information about transactions that have been undone.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name           | Data type | Description                                                 |
|-----------------------|-----------|-------------------------------------------------------------|
| userid                | integer   | ID of the user who generated the entry.                     |
| xact_id               | bigint    | ID for the undo transaction.                                |
| xact_id undone bigint |           | ID for the transaction that was undone.                     |
| undo_start_ts         | timestamp | Start time for the undo transaction.                        |
| undo_end_ts           | timestamp | End time for the undo transaction.                          |
| table_id              | bigint    | ID for the table that was affected by the undo transaction. |

## Sample query

To view a concise log of all undone transactions, type the following command:

```
select xact_id, xact_id undone, table_id from stl undone;
```

This command returns the following sample output:

```
xact_id | xact_id undone | table_id
-----+-----+-----
1344 | 1344 | 100192
1326 | 1326 | 100192
1551 | 1551 | 100192
(3 rows)
```

## STL\_UNIQUE

Analyzes execution steps that occur when a DISTINCT function is used in the SELECT list or when duplicates are removed in a UNION or INTERSECT query.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name  | Data type    | Description                                                                                                                                                                                                                                                                                                            |
|--------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid       | integer      | ID of the user who generated the entry.                                                                                                                                                                                                                                                                                |
| query        | integer      | Query ID. The query column can be used to join other system tables and views.                                                                                                                                                                                                                                          |
| slice        | integer      | Number that identifies the slice where the query was running.                                                                                                                                                                                                                                                          |
| segment      | integer      | Number that identifies the query segment.                                                                                                                                                                                                                                                                              |
| step         | integer      | Query step that ran.                                                                                                                                                                                                                                                                                                   |
| starttime    | timestamp    | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .                                                                                                                                     |
| endtime      | timestamp    | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .                                                                                                                                    |
| tasknum      | integer      | Number of the query task process that was assigned to run the step.                                                                                                                                                                                                                                                    |
| rows         | bigint       | Total number of rows that were processed.                                                                                                                                                                                                                                                                              |
| type         | character(6) | The type of step. Valid values are: <ul style="list-style-type: none"> <li>• HASHED. Indicates that the step used grouped, unsorted aggregation.</li> <li>• PLAIN. Indicates that the step used ungrouped, scalar aggregation.</li> <li>• SORTED. Indicates that the step used grouped, sorted aggregation.</li> </ul> |
| is_diskbased | character(1) | If true (t), the query was performed as a disk-based operation. If false (f), the query was performed in memory.                                                                                                                                                                                                       |

| Column name               | Data type | Description                                                             |
|---------------------------|-----------|-------------------------------------------------------------------------|
| slots                     | integer   | Total number of hash buckets.                                           |
| workmem                   | bigint    | Total number of bytes in working memory that were assigned to the step. |
| max_buffers_used          | bigint    | Maximum number of buffers used in the hash table before going to disk.  |
| resizes                   | integer   | This information is for internal use only.                              |
| occupied                  | integer   | This information is for internal use only.                              |
| flushable                 | integer   | This information is for internal use only.                              |
| used_unique_persistent(1) | integer   | This information is for internal use only.                              |
| bytes                     | bigint    | The number of bytes of all the output rows for the step.                |

## Sample queries

Suppose you run the following query:

```
select distinct eventname
from event
order by 1;
```

Assuming the ID for the previous query is 6313, the following example shows the number of rows produced by the unique step for each slice in segments 0 and 1.

```
select query, slice, segment, step, datediff(msec, starttime, endtime) as msec, tasknum,
rows
from stl_unique where query = 6313
order by query desc, slice, segment, step;
```

| query | slice | segment | step | msec | tasknum | rows |
|-------|-------|---------|------|------|---------|------|
| 6313  | 0     | 0       | 2    | 0    | 22      | 550  |
| 6313  | 0     | 1       | 1    | 256  | 20      | 145  |
| 6313  | 1     | 0       | 2    | 1    | 23      | 540  |
| 6313  | 1     | 1       | 1    | 42   | 21      | 127  |
| 6313  | 2     | 0       | 2    | 1    | 22      | 540  |
| 6313  | 2     | 1       | 1    | 255  | 20      | 158  |
| 6313  | 3     | 0       | 2    | 1    | 23      | 542  |
| 6313  | 3     | 1       | 1    | 38   | 21      | 146  |

(8 rows)

## STL\_UNLOAD\_LOG

Records the details for an unload operation.

STL\_UNLOAD\_LOG records one row for each file created by an UNLOAD statement. For example, if an UNLOAD creates 12 files, STL\_UNLOAD\_LOG will contain 12 corresponding rows.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name   | Data type       | Description                                                   |
|---------------|-----------------|---------------------------------------------------------------|
| userid        | integer         | ID of the user who generated the entry.                       |
| query         | integer         | The query ID.                                                 |
| slice         | integer         | Number that identifies the slice where the query was running. |
| pid           | integer         | Process ID associated with the query statement.               |
| path          | character(1280) | The complete Amazon S3 object path for the file.              |
| start_time    | timestamp       | Start time for the transaction.                               |
| end_time      | timestamp       | End time for the transaction.                                 |
| line_count    | bigint          | Number of lines (rows) unloaded to the file.                  |
| transfer_size | bigint          | Number of bytes transferred.                                  |
| file_format   | character(10)   | Format of unloaded file.                                      |

## Sample query

To get a list of the files that were written to Amazon S3 by an UNLOAD command, you can call an Amazon S3 list operation after the UNLOAD completes. You can also query `STL_UNLOAD_LOG`.

The following query returns the pathname for files that were created by an UNLOAD for the last query completed:

```
select query, substring(path,0,40) as path
from stl_unload_log
where query = pg_last_query_id()
order by path;
```

This command returns the following sample output:

| query    | path                             |
|----------|----------------------------------|
| 2320     | s3://my-bucket/venue0000_part_00 |
| 2320     | s3://my-bucket/venue0001_part_00 |
| 2320     | s3://my-bucket/venue0002_part_00 |
| 2320     | s3://my-bucket/venue0003_part_00 |
| (4 rows) |                                  |

## STL\_USAGE\_CONTROL

The `STL_USAGE_CONTROL` view contains information that is logged when a usage limit is reached. For more information about usage limits, see [Managing Usage Limits](#) in the *Amazon Redshift Management Guide*.

`STL_USAGE_CONTROL` is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name    | Data type     | Description                                                                                                            |
|----------------|---------------|------------------------------------------------------------------------------------------------------------------------|
| eventtime      | timestamp     | The time (UTC) when the query exceeded a usage limit.                                                                  |
| query          | integer       | The query identifier. You can use this ID to join various other system tables and views.                               |
| xid            | bigint        | The transaction identifier.                                                                                            |
| pid            | integer       | The process identifier associated with the query.                                                                      |
| usage_limit_id | character(40) | A universally unique identifier (UUID) generated by Amazon Redshift, for example 25d9297e-3e7b-41c8-9f4d-c4b6eb731c09. |
| feature_type   | character(30) | The feature whose usage limit was exceeded. Possible values include CONCURRENCY_SCALING and SPECTRUM.                  |

## Sample query

The following SQL example returns some of the information logged when a usage limit is reached.

```
select query, pid, eventtime, feature_type
from stl_usage_control
order by eventtime desc
limit 5;
```

## STL\_USERLOG

Records details for the following changes to a database user:

- Create user
- Drop user
- Alter user (rename)
- Alter user (alter properties)

This view is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type     | Description                                                                             |
|-------------|---------------|-----------------------------------------------------------------------------------------|
| userid      | integer       | ID of the user affected by the change.                                                  |
| username    | character(50) | User name of the user affected by the change.                                           |
| oldusername | character(50) | For a rename action, the original user name. For any other action, this field is empty. |

| Column name | Data type     | Description                                                                                                                                  |
|-------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| action      | character(10) | Action that occurred. Valid values: <ul style="list-style-type: none"><li>• Alter</li><li>• Create</li><li>• Drop</li><li>• Rename</li></ul> |
| usecreatedb | integer       | If true (1), indicates that the user has create database privileges.                                                                         |
| usesuper    | integer       | If true (1), indicates that the user is a superuser.                                                                                         |
| usecatupd   | integer       | If true (1), indicates that the user can update system catalogs.                                                                             |
| valuntil    | timestamp     | Password expiration date.                                                                                                                    |
| pid         | integer       | Process ID.                                                                                                                                  |
| xid         | bigint        | Transaction ID.                                                                                                                              |
| recordtime  | timestamp     | Time in UTC that the query started.                                                                                                          |

## Sample queries

The following example performs four user actions, then queries the STL\_USERLOG view.

```
create user userlog1 password 'Userlog1';
alter user userlog1 createdb createuser;
alter user userlog1 rename to userlog2;
drop user userlog2;

select userid, username, oldusername, action, usecreatedb, usesuper from stl_userlog order
by recordtime desc;
```

| userid | username | oldusername | action | usecreatedb | usesuper |
|--------|----------|-------------|--------|-------------|----------|
| 108    | userlog2 |             | drop   | 1           | 1        |
| 108    | userlog2 | userlog1    | rename | 1           | 1        |
| 108    | userlog1 |             | alter  | 1           | 1        |
| 108    | userlog1 |             | create | 0           | 0        |

(4 rows)

## STL.UtilityText

Captures the text of non-SELECT SQL commands run on the database.

Query the STL.UtilityText view to capture the following subset of SQL statements that were run on the system:

- ABORT, BEGIN, COMMIT, END, ROLLBACK

- ANALYZE
- CALL
- CANCEL
- COMMENT
- CREATE, ALTER, DROP DATABASE
- CREATE, ALTER, DROP USER
- EXPLAIN
- GRANT, REVOKE
- LOCK
- RESET
- SET
- SHOW
- TRUNCATE

See also [STL\\_DDLTEXT \(p. 1507\)](#), [STL\\_QUERYTEXT \(p. 1544\)](#), and [SVL\\_STATEMENTTEXT \(p. 1661\)](#).

Use the STARTTIME and ENDTIME columns to find out which statements were logged during a given time period. Long blocks of SQL text are broken into lines 200 characters long; the SEQUENCE column identifies fragments of text that belong to a single statement.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type      | Description                                                                                                                                                                                       |
|-------------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer        | ID of the user who generated the entry.                                                                                                                                                           |
| xid         | bigint         | Transaction ID.                                                                                                                                                                                   |
| pid         | integer        | Process ID associated with the query statement.                                                                                                                                                   |
| label       | character(320) | Either the name of the file used to run the query or a label defined with a SET QUERY_GROUP command. If the query is not file-based or the QUERY_GROUP parameter is not set, this field is blank. |
| starttime   | timestamp      | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .                |
| endtime     | timestamp      | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .               |
| sequence    | integer        | When a single statement contains more than 200 characters, additional rows are logged for that statement. Sequence 0 is the first row, 1 is the second, and so on.                                |
| text        | character(200) | SQL text, in 200-character increments. This field might contain special characters such as backslash (\) and newline (\n).                                                                        |

## Sample queries

The following query returns the text for "utility" commands that were run on January 26th, 2012. In this case, some SET commands and a SHOW ALL command were run:

```
select starttime, sequence, rtrim(text)
from stl.utilitytext
where starttime like '2012-01-26%'
order by starttime, sequence;
```

| starttime                  | sequence | rtrim                               |
|----------------------------|----------|-------------------------------------|
| 2012-01-26 13:05:52.529235 | 0        | show all;                           |
| 2012-01-26 13:20:31.660255 | 0        | SET query_group to ''               |
| 2012-01-26 13:20:54.956131 | 0        | SET query_group to 'soldunsold.sql' |
| ...                        |          |                                     |

## Reconstructing Stored SQL

To reconstruct the SQL stored in the `text` column of `STL.UtilityText`, run a `SELECT` statement to create SQL from 1 or more parts in the `text` column. Before running the reconstructed SQL, replace any `(\n)` special characters with a new line. The result of the following `SELECT` statement is rows of reconstructed SQL in the `query_statement` field.

```
SELECT LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END) WITHIN GROUP
 (ORDER BY sequence) as query_statement
FROM stl_utilitytext GROUP BY xid order by xid;
```

For example, the following query sets the query\_group to a string of zeros. The query itself is longer than 200 characters and is stored in parts in STL.utilityText.

In this example, the query is stored in 2 parts (rows) in the `text` column of `STL.utilityText`.

```
select query, sequence, text
from stl_utilitytext where query=pg_last_query_id() order by query desc, sequence limit 10;
```

To reconstruct the SQL stored in `STL_UNUTILITYTEXT`, run the following SQL.

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '') within
group (order by sequence) AS query_statement
from stl_utilitytext where query=pg_last_query_id();
```

To use the resulting reconstructed SQL in your client, replace any (\n) special characters with a new line.

## query\_statement

| Column name | Data type     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer       | The ID of the user who generated the entry.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| xid         | bigint        | The transaction ID for the VACUUM statement. You can join this table to the STL_QUERY view to see the individual SQL statements that are run for a given VACUUM transaction. If you vacuum the whole database, each table is vacuumed in a separate transaction.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| table_id    | integer       | The Table ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| status      | character(30) | <p>The status of the VACUUM operation for each table. Possible values are the following:</p> <ul style="list-style-type: none"><li>• <b>Started</b></li><li>• <b>Started Delete Only</b></li><li>• <b>Started Delete Only (Sorted &gt;= nn%)</b></li></ul> <p>Only the delete phase was started for a VACUUM FULL. The sort phase was skipped because the table was already sorted at or above the sort threshold.</p> <ul style="list-style-type: none"><li>• <b>Started Sort Only</b></li><li>• <b>Started Ranged Partition</b></li><li>• <b>Started Reindex</b></li><li>• <b>Finished</b></li></ul> <p>Time the operation completed for the table. To find out how long a vacuum operation took on a specific table, subtract the Started time from the Finished time for a particular transaction ID and table ID.</p> <ul style="list-style-type: none"><li>• <b>Skipped</b></li></ul> |

| Column name | Data type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|             |           | <p>The table was skipped because the table was fully sorted and no rows were marked for deletion.</p> <ul style="list-style-type: none"> <li>• <b>Skipped (delete only)</b></li> </ul> <p>The table was skipped because DELETE ONLY was specified and no rows were marked for deletion.</p> <ul style="list-style-type: none"> <li>• <b>Skipped (sort only)</b></li> </ul> <p>The table was skipped because SORT ONLY was specified and the table was already sorted fully sorted.</p> <ul style="list-style-type: none"> <li>• <b>Skipped (sort only, sorted&gt;=xx%)</b></li> </ul> <p>The table was skipped because SORT ONLY was specified and the table was already sorted at or above the sort threshold.</p> <ul style="list-style-type: none"> <li>• <b>Skipped (0 rows)</b></li> </ul> <p>The table was skipped because it was empty.</p> <ul style="list-style-type: none"> <li>• <b>VacuumBG</b></li> </ul> <p>An automatic vacuum operation was performed in the background. This status is prepended to other statuses when they're performed automatically. For example, a delete only vacuum performed automatically would have a starting row with the status [VacuumBG] Started Delete Only.</p> <p>For more information about the VACUUM sort threshold setting, see <a href="#">VACUUM (p. 1015)</a>.</p> |
| rows        | bigint    | The actual number of rows in the table plus any deleted rows that are still stored on disk (waiting to be vacuumed). This column shows the count before the vacuum started for rows with a <b>Started</b> status, and the count after the vacuum for rows with a <b>Finished</b> status.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| sortedrows  | integer   | The number of rows in the table that are sorted. This column shows the count before the vacuum started for rows with <b>Started</b> in the Status column, and the count after the vacuum for rows with <b>Finished</b> in the Status column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| blocks      | integer   | The total number of data blocks used to store the table data before the vacuum operation (rows with a <b>Started</b> status) and after the vacuum operation ( <b>Finished</b> column). Each data block uses 1 MB.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

| Column name          | Data type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| max_merge_partitions | integer   | This column is used for performance analysis and represents the maximum number of partitions that vacuum can process for the table per merge phase iteration. (Vacuum sorts the unsorted region into one or more sorted partitions. Depending on the number of columns in the table and the current Amazon Redshift configuration, the merge phase can process a maximum number of partitions in a single merge iteration. The merge phase will still work if the number of sorted partitions exceeds the maximum number of merge partitions, but more merge iterations will be required.) |
| eventtime            | timestamp | When the vacuum operation started or finished.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| reclaimable_rows     | bigint    | The number of reclaimable rows for the current cutoff_xid. This column shows Redshift's estimated number of reclaimable rows before the vacuum started for rows with a <b>Started</b> status, and the actual number of reclaimable rows remaining after the vacuum for rows with a <b>Finished</b> status.                                                                                                                                                                                                                                                                                 |
| reclaimable_space_mb | bigint    | Reclaimable space in MB for the current cutoff_xid. This column shows Redshift's estimated amount of reclaimable space before the vacuum started for rows with a <b>Started</b> status, and the actual amount of reclaimable space remaining after the vacuum for rows with a <b>Finished</b> status.                                                                                                                                                                                                                                                                                      |
| cutoff_xid           | bigint    | The cutoff transaction ID for the VACUUM operation. Any transactions after the cutoff are not included in the VACUUM operation.                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| is_recluster         | integer   | If 1 (true), the VACUUM operation executed the recluster algorithm, If 0 (false), it was not.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

## Sample queries

The following query reports vacuum statistics for table 108313. The table was vacuumed following a series of inserts and deletes.

```
select xid, table_id, status, rows, sortedrows, blocks, eventtime,
 reclaimable_rows, reclaimable_space_mb
 from stl_vacuum where table_id=108313 order by eventtime;

 xid | table_id | status | rows | sortedrows | blocks | eventtime
 | | Started | | | | |
 | | reclaimable_rows | | | | |
-----+-----+-----+-----+-----+-----+-----+
 14294 | 108313 | Started | 1950 | 408 | 28 | 2016-05-19
 17:36:01 | | 984 | 17 | |
 14294 | 108313 | Finished | 966 | 966 | 11 | 2016-05-19
 18:26:13 | | 0 | 0 | |
 15126 | 108313 | Skipped(sorted>=95%) | 966 | 966 | 11 | 2016-05-19
 18:26:38 | | 0 | 0 | |
```

At the start of the VACUUM, the table contained 1,950 rows stored in 28 1 MB blocks. Amazon Redshift estimated it could reclaim 984, or 17 blocks of disk space, with a vacuum operation.

In the row for the Finished status, the ROWS column shows a value of 966, and the BLOCKS column value is 11, down from 28. The vacuum reclaimed the estimated amount of disk space, with no reclaimable rows or space remaining after the vacuum operation completed.

In the sort phase (transaction 15126), the vacuum was able to skip the table because the rows were inserted in sort key order.

The following example shows the statistics for a SORT ONLY vacuum on the SALES table (table 110116 in this example) after a large INSERT operation:

```
vacuum sort only sales;

select xid, table_id, status, rows, sortedrows, blocks, eventtime
from stl_vacuum order by xid, table_id, eventtime;

xid |table_id| status | rows |sortedrows|blocks| eventtime
----+-----+-----+-----+-----+-----+
...
2925| 110116 |Started Sort Only|1379648| 172456 | 132 | 2011-02-24 16:25:21...
2925| 110116 |Finished |1379648| 1379648 | 132 | 2011-02-24 16:26:28...
```

## STL\_WINDOW

Analyzes query steps that perform window functions.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                                                                                                                             |
| query       | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                       |
| slice       | integer   | Number that identifies the slice where the query was running.                                                                                                                       |
| segment     | integer   | Number that identifies the query segment.                                                                                                                                           |
| step        | integer   | Query step that ran.                                                                                                                                                                |
| starttime   | timestamp | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> .  |
| endtime     | timestamp | Time in UTC that the query finished. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |
| tasknum     | integer   | Number of the query task process that was assigned to run the step.                                                                                                                 |

| Column name  | Data type    | Description                                                                                                      |
|--------------|--------------|------------------------------------------------------------------------------------------------------------------|
| rows         | bigint       | Total number of rows that were processed.                                                                        |
| is_diskbased | character(1) | If true (t), the query was performed as a disk-based operation. If false (f), the query was performed in memory. |
| workmem      | bigint       | Total number of bytes in working memory that were assigned to the step.                                          |

## Sample queries

The following example returns window function results for slice 0 and segment 3.

```
select query, tasknum, rows, is_diskbased, workmem
from stl_window
where slice=0 and segment=3;
```

| query    | tasknum | rows | is_diskbased | workmem  |
|----------|---------|------|--------------|----------|
| 86326    | 36      | 1857 | f            | 95256616 |
| 705      | 15      | 1857 | f            | 95256616 |
| 86399    | 27      | 1857 | f            | 95256616 |
| 649      | 10      | 0    | f            | 95256616 |
| (4 rows) |         |      |              |          |

## STL\_WLM\_ERROR

Records all WLM-related errors as they occur.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name  | Data type      | Description                                  |
|--------------|----------------|----------------------------------------------|
| userid       | integer        | ID of the user who generated the entry.      |
| recordtime   | timestamp      | Time that the error occurred.                |
| pid          | integer        | ID for the process that generated the error. |
| error_string | character(256) | Error description.                           |

## STL\_WLM\_RULE\_ACTION

Records details about actions resulting from WLM query monitoring rules associated with user-defined queues. For more information, see [WLM query monitoring rules \(p. 527\)](#).

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name        | Data type      | Description                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid             | integer        | User that ran the query.                                                                                                                                                                                                                                                                                                                                                              |
| query              | integer        | Query ID.                                                                                                                                                                                                                                                                                                                                                                             |
| service_class      | integer        | ID for the service class. Query queues are defined in the WLM configuration. Service classes greater than 5 are user-defined queues.                                                                                                                                                                                                                                                  |
| rule               | character(256) | Name of a query monitoring rule.                                                                                                                                                                                                                                                                                                                                                      |
| action             | character(256) | <p>Resulting action. Possible values are as follows:</p> <ul style="list-style-type: none"> <li>• log</li> <li>• hop(reassign)</li> <li>• hop(restart)</li> <li>• abort</li> <li>• change_query_priority</li> <li>• none</li> </ul> <p>A value of none indicates that the rule's predicates were met but the action was superseded by another rule with a higher severity action.</p> |
| recordtime         | timestamp      | Time the action was logged.                                                                                                                                                                                                                                                                                                                                                           |
| action_value       | character(256) | If action is change_query_priority, then possible values are highest, high, normal, low, and lowest.<br><br>If action is log, hop, or abort then the value is empty.                                                                                                                                                                                                                  |
| service_class_name | character(64)  | The name of the service class.                                                                                                                                                                                                                                                                                                                                                        |

## Sample queries

The following example finds queries that were stopped by a query monitoring rule.

```
Select query, rule
from stl_wlm_rule_action
where action = 'abort'
order by query;
```

## STL\_WLM\_QUERY

Contains a record of each attempted execution of a query in a service class handled by WLM.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name              | Data type     | Description                                                                                                                                                                            |
|--------------------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid                   | integer       | ID of the user who generated the entry.                                                                                                                                                |
| xid                      | integer       | Transaction ID of the query or subquery.                                                                                                                                               |
| task                     | integer       | ID used to track a query through the workload manager. Can be associated with multiple query IDs. If a query is restarted, the query is assigned a new query ID but not a new task ID. |
| query                    | integer       | Query ID. If a query is restarted, the query is assigned a new query ID but not a new task ID.                                                                                         |
| service_class            | integer       | ID for the service class. For a list of service class IDs, see <a href="#">WLM service class IDs (p. 534)</a> .                                                                        |
| slot_count               | integer       | Number of WLM query slots.                                                                                                                                                             |
| service_class_start_time | timestamp     | Time that the query was assigned to the service class.                                                                                                                                 |
| queue_start_time         | timestamp     | Time that the query entered the queue for the service class.                                                                                                                           |
| queue_end_time           | timestamp     | Time when the query left the queue for the service class.                                                                                                                              |
| total_queue_time         | bigint        | Total number of microseconds that the query spent in the queue.                                                                                                                        |
| exec_start_time          | timestamp     | Time that the query began executing in the service class.                                                                                                                              |
| exec_end_time            | timestamp     | Time that the query completed execution in the service class.                                                                                                                          |
| total_exec_time          | bigint        | Number of microseconds that the query spent executing.                                                                                                                                 |
| service_class_end_time   | timestamp     | Time that the query left the service class.                                                                                                                                            |
| final_state              | character(16) | Reserved for system use.                                                                                                                                                               |
| est_peak_mem             | bigint        | Reserved for system use.                                                                                                                                                               |
| query_priority           | char(20)      | The priority of the query. Possible values are n/a, lowest, low, normal, high, and highest, where n/a means that query priority isn't supported.                                       |
| service_class_name       | character(64) | The service class name. For more information about service classes, see <a href="#">WLM system tables and views</a> .                                                                  |

## Sample queries

### View average query Time in queues and executing

The following queries display the current configuration for service classes greater than 4. For a list of service class IDs, see [WLM service class IDs \(p. 534\)](#).

The following query returns the average time (in microseconds) that each query spent in query queues and executing for each service class.

```
select service_class as svc_class, count(*),
avg(datediff(microseconds, queue_start_time, queue_end_time)) as avg_queue_time,
avg(datediff(microseconds, exec_start_time, exec_end_time)) as avg_exec_time
from stl_wlm_query
where service_class > 4
group by service_class
order by service_class;
```

This query returns the following sample output:

| svc_class | count | avg_queue_time | avg_exec_time |
|-----------|-------|----------------|---------------|
| 5         | 20103 | 0              | 80415         |
| 5         | 3421  | 34015          | 234015        |
| 6         | 42    | 0              | 944266        |
| 7         | 196   | 6439           | 1364399       |

(4 rows)

### **View maximum query time in queues and executing**

The following query returns the maximum amount of time (in microseconds) that a query spent in any query queue and executing for each service class.

```
select service_class as svc_class, count(*),
max(datediff(microseconds, queue_start_time, queue_end_time)) as max_queue_time,
max(datediff(microseconds, exec_start_time, exec_end_time)) as max_exec_time
from stl_wlm_query
where svc_class > 5
group by service_class
order by service_class;
```

| svc_class | count | max_queue_time | max_exec_time |
|-----------|-------|----------------|---------------|
| 6         | 42    | 0              | 3775896       |
| 7         | 197   | 37947          | 16379473      |

(4 rows)

## **STV tables for snapshot data**

STV tables are virtual system tables that contain snapshots of the current system data.

### **Topics**

- [STV\\_ACTIVE\\_CURSORS \(p. 1578\)](#)
- [STV\\_BLOCKLIST \(p. 1579\)](#)
- [STV\\_CURSOR\\_CONFIGURATION \(p. 1581\)](#)
- [STV\\_DB\\_ISOLATION\\_LEVEL \(p. 1582\)](#)
- [STV\\_EXEC\\_STATE \(p. 1582\)](#)
- [STV\\_INFLIGHT \(p. 1583\)](#)
- [STV\\_LOAD\\_STATE \(p. 1585\)](#)
- [STV\\_LOCKS \(p. 1586\)](#)

- [STV\\_ML\\_MODEL\\_INFO \(p. 1587\)](#)
- [STV\\_MV\\_DEPS \(p. 1587\)](#)
- [STV\\_MV\\_INFO \(p. 1588\)](#)
- [STV\\_NODE\\_STORAGE\\_CAPACITY \(p. 1590\)](#)
- [STV\\_PARTITIONS \(p. 1591\)](#)
- [STV\\_QUERY\\_METRICS \(p. 1592\)](#)
- [STV\\_RECENTS \(p. 1596\)](#)
- [STV\\_SESSIONS \(p. 1597\)](#)
- [STV\\_SLICES \(p. 1598\)](#)
- [STV\\_STARTUP\\_RECOVERY\\_STATE \(p. 1599\)](#)
- [STV\\_TBL\\_PERM \(p. 1599\)](#)
- [STV\\_TBL\\_TRANS \(p. 1601\)](#)
- [STV\\_WLM\\_QMR\\_CONFIG \(p. 1603\)](#)
- [STV\\_WLM\\_CLASSIFICATION\\_CONFIG \(p. 1603\)](#)
- [STV\\_WLM\\_QUERY\\_QUEUE\\_STATE \(p. 1604\)](#)
- [STV\\_WLM\\_QUERY\\_STATE \(p. 1605\)](#)
- [STV\\_WLM\\_QUERY\\_TASK\\_STATE \(p. 1607\)](#)
- [STV\\_WLM\\_SERVICE\\_CLASS\\_CONFIG \(p. 1608\)](#)
- [STV\\_WLM\\_SERVICE\\_CLASS\\_STATE \(p. 1609\)](#)

## STV\_ACTIVE\_CURSORS

STV\_ACTIVE\_CURSORS displays details for currently open cursors. For more information, see [DECLARE \(p. 855\)](#).

STV\_ACTIVE\_CURSORS is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#). A user can only view cursors opened by that user. A superuser can view all cursors.

### Table columns

| Column name  | Data type      | Description                                                  |
|--------------|----------------|--------------------------------------------------------------|
| userid       | integer        | ID of user who generated entry.                              |
| name         | character(256) | cursor name.                                                 |
| xid          | bigint         | Transaction context.                                         |
| pid          | integer        | Leader process running the query.                            |
| starttime    | timestamp      | Time when the cursor was declared.                           |
| row_count    | bigint         | Number of rows in the cursor result set.                     |
| byte_count   | bigint         | Number of bytes in the cursor result set.                    |
| fetched_rows | bigint         | Number of rows currently fetched from the cursor result set. |

## STV\_BLOCKLIST

STV\_BLOCKLIST contains the number of 1 MB disk blocks that are used by each slice, table, or column in a database.

Use aggregate queries with STV\_BLOCKLIST, as the following examples show, to determine the number of 1 MB disk blocks allocated per database, table, slice, or column. You can also use [STV\\_PARTITIONS \(p. 1591\)](#) to view summary information about disk utilization.

STV\_BLOCKLIST is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name     | Data type | Description                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| slice           | integer   | Node slice.                                                                                                                                                                                                                                                                                                                                                                            |
| col             | integer   | Zero-based index for the column. Every table you create has three hidden columns appended to it: INSERT_XID, DELETE_XID, and ROW_ID (OID). A table with 3 user-defined columns contains 6 actual columns, and the user-defined columns are internally numbered as 0, 1, and 2. The INSERT_XID, DELETE_XID, and ROW_ID columns are numbered 3, 4, and 5, respectively, in this example. |
| tbl             | integer   | Table ID for the database table.                                                                                                                                                                                                                                                                                                                                                       |
| blocknum        | integer   | ID for the data block.                                                                                                                                                                                                                                                                                                                                                                 |
| num_values      | integer   | Number of values contained on the block.                                                                                                                                                                                                                                                                                                                                               |
| extended_limits | integer   | For internal use.                                                                                                                                                                                                                                                                                                                                                                      |
| minvalue        | bigint    | Minimum data value of the block. Stores first eight characters as 64-bit integer for non-numeric data. Used for disk scanning.                                                                                                                                                                                                                                                         |
| maxvalue        | bigint    | Maximum data value of the block. Stores first eight characters as 64-bit integer for non-numeric data. Used for disk scanning.                                                                                                                                                                                                                                                         |
| sb_pos          | integer   | Internal Amazon Redshift identifier for super block position on the disk.                                                                                                                                                                                                                                                                                                              |
| pinned          | integer   | Whether or not the block is pinned into memory as part of pre-load. 0 = false; 1 = true. Default is false.                                                                                                                                                                                                                                                                             |
| on_disk         | integer   | Whether or not the block is automatically stored on disk. 0 = false; 1 = true. Default is false.                                                                                                                                                                                                                                                                                       |
| modified        | integer   | Whether or not the block has been modified. 0 = false; 1 = true. Default is false.                                                                                                                                                                                                                                                                                                     |
| hdr_modified    | integer   | Whether or not the block header has been modified. 0 = false; 1 = true. Default is false.                                                                                                                                                                                                                                                                                              |
| unsorted        | integer   | Whether or not a block is unsorted. 0 = false; 1 = true. Default is true.                                                                                                                                                                                                                                                                                                              |
| tombstone       | integer   | For internal use.                                                                                                                                                                                                                                                                                                                                                                      |
| preferred_disk  | integer   | Disk number that the block should be on, unless the disk has failed. Once the disk has been fixed, the block will move back to this disk.                                                                                                                                                                                                                                              |

| Column name | Data type | Description                                                                                                                                            |
|-------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| temporary   | integer   | Whether or not the block contains temporary data, such as from a temporary table or intermediate query results. 0 = false; 1 = true. Default is false. |
| newblock    | integer   | Indicates whether or not a block is new (true) or was never committed to disk (false). 0 = false; 1 = true.                                            |
| num_readers | integer   | Number of references on each block.                                                                                                                    |
| flags       | integer   | Internal Amazon Redshift flags for the block header.                                                                                                   |

## Sample queries

STV\_BLOCKLIST contains one row per allocated disk block, so a query that selects all the rows potentially returns a very large number of rows. We recommend using only aggregate queries with STV\_BLOCKLIST.

The [SVV\\_DISKUSAGE \(p. 1426\)](#) view provides similar information in a more user-friendly format; however, the following example demonstrates one use of the STV\_BLOCKLIST table.

To determine the number of 1 MB blocks used by each column in the VENUE table, type the following query:

```
select col, count(*)
from stv_blocklist, stv_tbl_perm
where stv_blocklist.tbl = stv_tbl_perm.id
and stv_blocklist.slice = stv_tbl_perm.slice
and stv_tbl_perm.name = 'venue'
group by col
order by col;
```

This query returns the number of 1 MB blocks allocated to each column in the VENUE table, shown by the following sample data:

| col      | count |
|----------|-------|
| 0        | 4     |
| 1        | 4     |
| 2        | 4     |
| 3        | 4     |
| 4        | 4     |
| 5        | 4     |
| 7        | 4     |
| 8        | 4     |
| (8 rows) |       |

The following query shows whether or not table data is actually distributed over all slices:

```
select trim(name) as table, stv_blocklist.slice, stv_tbl_perm.rows
from stv_blocklist, stv_tbl_perm
where stv_blocklist.tbl=stv_tbl_perm.id
and stv_tbl_perm.slice=stv_blocklist.slice
and stv_blocklist.id > 10000 and name not like '%#m%'
and name not like 'systable%'
group by name, stv_blocklist.slice, stv_tbl_perm.rows
order by 3 desc;
```

This query produces the following sample output, showing the even data distribution for the table with the most rows:

| table      | slice | rows  |
|------------|-------|-------|
| listing    | 13    | 10527 |
| listing    | 14    | 10526 |
| listing    | 8     | 10526 |
| listing    | 9     | 10526 |
| listing    | 7     | 10525 |
| listing    | 4     | 10525 |
| listing    | 17    | 10525 |
| listing    | 11    | 10525 |
| listing    | 5     | 10525 |
| listing    | 18    | 10525 |
| listing    | 12    | 10525 |
| listing    | 3     | 10525 |
| listing    | 10    | 10525 |
| listing    | 2     | 10524 |
| listing    | 15    | 10524 |
| listing    | 16    | 10524 |
| listing    | 6     | 10524 |
| listing    | 19    | 10524 |
| listing    | 1     | 10523 |
| listing    | 0     | 10521 |
| ...        |       |       |
| (180 rows) |       |       |

The following query determines whether any tombstoned blocks were committed to disk:

| select slice, col, tbl, blocknum, newblock<br>from stv_blocklist<br>where tombstone > 0; |     |        |          |          |
|------------------------------------------------------------------------------------------|-----|--------|----------|----------|
| slice                                                                                    | col | tbl    | blocknum | newblock |
| 4                                                                                        | 0   | 101285 | 0        | 1        |
| 4                                                                                        | 2   | 101285 | 0        | 1        |
| 4                                                                                        | 4   | 101285 | 1        | 1        |
| 5                                                                                        | 2   | 101285 | 0        | 1        |
| 5                                                                                        | 0   | 101285 | 0        | 1        |
| 5                                                                                        | 1   | 101285 | 0        | 1        |
| 5                                                                                        | 4   | 101285 | 1        | 1        |
| ...                                                                                      |     |        |          |          |
| (24 rows)                                                                                |     |        |          |          |

## STV\_CURSOR\_CONFIGURATION

STV\_CURSOR\_CONFIGURATION displays cursor configuration constraints. For more information, see [Cursor constraints \(p. 856\)](#).

STV\_CURSOR\_CONFIGURATION is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name    | Data type | Description                       |
|----------------|-----------|-----------------------------------|
| current_cursor | integer   | Number of cursors currently open. |

| Column name       | Data type | Description                                                                                                                               |
|-------------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------|
| max_diskspace     | integer   | Amount of disk space available for cursors, in megabytes. This constraint is based on the maximum cursor result set size for the cluster. |
| current_diskspace | integer   | Amount of disk space currently used by cursors, in megabytes.                                                                             |

## STV\_DB\_ISOLATION\_LEVEL

STV\_DB\_ISOLATION\_LEVEL displays the current isolation level for databases. For more information about isolation levels, see [CREATE DATABASE \(p. 745\)](#).

STV\_DB\_ISOLATION\_LEVEL is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name     | Data type      | Description                                                                                   |
|-----------------|----------------|-----------------------------------------------------------------------------------------------|
| db_name         | character(128) | database name.                                                                                |
| isolation_level | character(20)  | isolation level of the database. Possible values include Serializable and Snapshot Isolation. |

## STV\_EXEC\_STATE

Use the STV\_EXEC\_STATE table to find out information about queries and query steps that are actively running on compute nodes.

This information is usually used only to troubleshoot engineering issues. The views SVV\_QUERY\_STATE and SVL\_QUERY\_SUMMARY extract their information from STV\_EXEC\_STATE.

STV\_EXEC\_STATE is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                  |
|-------------|-----------|----------------------------------------------------------------------------------------------|
| userid      | integer   | ID of user who generated entry.                                                              |
| query       | integer   | Query ID. Can be used to join various other system tables and views.                         |
| slice       | integer   | Node slice where the step completed.                                                         |
| segment     | integer   | Segment of the query that ran. A query segment is a series of steps.                         |
| step        | integer   | Step of the query segment that completed. A step is the smallest unit that a query performs. |

| Column name     | Data type | Description                                                                                                                                                                                                                                                                                                 |
|-----------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| starttime       | timestamp | Time that the step ran.                                                                                                                                                                                                                                                                                     |
| currenttime     | timestamp | Current time.                                                                                                                                                                                                                                                                                               |
| tasknum         | integer   | Query task process that is assigned to complete the step.                                                                                                                                                                                                                                                   |
| rows            | bigint    | Number of rows processed.                                                                                                                                                                                                                                                                                   |
| bytes           | bigint    | Number of bytes processed.                                                                                                                                                                                                                                                                                  |
| label           | char(256) | Step label, which consists of a query step name and, when applicable, table ID and table name (for example, scan <code>tbl=100448 name =user</code> ). Three-digit table IDs usually refer to scans of transient tables. When you see <code>tbl=0</code> , it usually refers to a scan of a constant value. |
| is_diskbased    | char(1)   | Whether this step of the query was completed as a disk-based operation: true ( <b>t</b> ) or false ( <b>f</b> ). Only certain steps, such as hash, sort, and aggregate steps, can go to disk. Many types of steps are always completed in memory.                                                           |
| workmem         | bigint    | Number of bytes of working memory assigned to the step.                                                                                                                                                                                                                                                     |
| num_parts       | integer   | Number of partitions a hash table is divided into during a hash step. A positive number in this column does not imply that the hash step ran as a disk-based operation. Check the value in the IS_DISKBASED column to see if the hash step was disk-based.                                                  |
| is_rrscan       | char(1)   | If true ( <b>t</b> ), indicates that range-restricted scan was used on the step. Default is false ( <b>f</b> ).                                                                                                                                                                                             |
| is_delayed_scan | char(1)   | If true ( <b>t</b> ), indicates that delayed scan was used on the step. Default is false ( <b>f</b> ).                                                                                                                                                                                                      |

## Sample queries

Rather than querying STV\_EXEC\_STATE directly, Amazon Redshift recommends querying SVL\_QUERY\_SUMMARY or SVV\_QUERY\_STATE to obtain the information in STV\_EXEC\_STATE in a more user-friendly format. See the [SVL\\_QUERY\\_SUMMARY \(p. 1647\)](#) or [SVV\\_QUERY\\_STATE \(p. 1442\)](#) table documentation for more details.

## STV\_INFLIGHT

Use the STV\_INFLIGHT table to determine what queries are currently running on the cluster. STV\_INFLIGHT does not show leader-node only queries. For more information, see [Leader node-only functions \(p. 1020\)](#).

STV\_INFLIGHT is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name        | Data type      | Description                                                                                                                                                                                                                                   |
|--------------------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid             | integer        | ID of user who generated entry.                                                                                                                                                                                                               |
| slice              | integer        | Slice where the query is running.                                                                                                                                                                                                             |
| query              | integer        | Query ID. Can be used to join various other system tables and views.                                                                                                                                                                          |
| label              | character(320) | Either the name of the file used to run the query or a label defined with a SET QUERY_GROUP command. If the query is not file-based or the QUERY_GROUP parameter is not set, this field is blank.                                             |
| xid                | bigint         | Transaction ID.                                                                                                                                                                                                                               |
| pid                | integer        | Process ID. All of the queries in a session are run in the same process, so this value remains constant if you run a series of queries in the same session. You can use this column to join to the <a href="#">STL_ERROR (p. 1513)</a> table. |
| starttime          | timestamp      | Time that the query started.                                                                                                                                                                                                                  |
| text               | character(100) | Query text, truncated to 100 characters if the statement exceeds that limit.                                                                                                                                                                  |
| suspended          | integer        | Whether the query is suspended or not. 0 = false; 1 = true.                                                                                                                                                                                   |
| insert_pristine    | integer        | Whether write queries are/were able to run while the current query is/was running. 1 = no write queries allowed. 0 = write queries allowed. This column is intended for use in debugging.                                                     |
| concurrency_status | integer        | Indicates whether the query ran on the main cluster or on a concurrency scaling cluster, Possible values are as follows:<br><br>0 - Ran on the main cluster<br><br>1 - Ran on a concurrency scaling cluster                                   |

## Sample queries

To view all active queries currently running on the database, type the following query:

```
select * from stv_inflight;
```

The sample output below shows two queries currently running, including the STV\_INFLIGHT query itself and a query that was run from a script called avgwait.sql:

```
select slice, query, trim(label) querylabel, pid,
starttime, substring(text,1,20) querytext
from stv_inflight;

slice|query|querylabel | pid | starttime | querytext
-----+-----+-----+-----+-----+-----+
```

|                         |  |    |  |             |  |                            |  |                            |  |                      |
|-------------------------|--|----|--|-------------|--|----------------------------|--|----------------------------|--|----------------------|
| 1011                    |  | 21 |  | 646         |  | 2012-01-26 13:23:15.645503 |  | select slice, query,       |  |                      |
| 1011                    |  | 20 |  | avgwait.sql |  | 499                        |  | 2012-01-26 13:23:14.159912 |  | select avg(datediff( |
| <small>(2 rows)</small> |  |    |  |             |  |                            |  |                            |  |                      |

## STV\_LOAD\_STATE

Use the STV\_LOAD\_STATE table to find information about current state of ongoing COPY statements.

The COPY command updates this table after every million records are loaded.

STV\_LOAD\_STATE is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name              | Data type      | Description                                                                                                                                                 |
|--------------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid                   | integer        | ID of user who generated entry.                                                                                                                             |
| session                  | integer        | Session PID of process doing the load.                                                                                                                      |
| query                    | integer        | Query ID. Can be used to join various other system tables and views.                                                                                        |
| slice                    | integer        | Node slice number.                                                                                                                                          |
| pid                      | integer        | Process ID. All of the queries in a session are run in the same process, so this value remains constant if you run a series of queries in the same session. |
| recordtime               | timestamp      | Time the record is logged.                                                                                                                                  |
| bytes_to_load            | bigint         | Total number of bytes to be loaded by this slice. This is 0 if the data being loaded is compressed                                                          |
| bytes_loaded             | bigint         | Number of bytes loaded by this slice. If the data being loaded is compressed, this is the number of bytes loaded after the data is uncompressed.            |
| bytes_to_load_compressed | bigint         | Total number of bytes of compressed data to be loaded by this slice. This is 0 if the data being loaded is not compressed.                                  |
| bytes_loaded_compressed  | bigint         | Number of bytes of compressed data loaded by this slice. This is 0 if the data being loaded is not compressed.                                              |
| lines                    | integer        | Number of lines loaded by this slice.                                                                                                                       |
| num_files                | integer        | Number of files to be loaded by this slice.                                                                                                                 |
| num_files_complete       | integer        | Number of files loaded by this slice.                                                                                                                       |
| current_file             | character(256) | Name of the file being loaded by this slice.                                                                                                                |
| pct_complete             | integer        | Percentage of data load completed by this slice.                                                                                                            |

### Sample query

To view the progress of each slice for a COPY command, type the following query. This example uses the PG\_LAST\_COPY\_ID() function to retrieve information for the last COPY command.

```
select slice , bytes_loaded, bytes_to_load , pct_complete from stv_load_state where query =
pg_last_copy_id();

slice | bytes_loaded | bytes_to_load | pct_complete
-----+-----+-----+-----+
 2 | 0 | 0 | 0
 3 | 12840898 | 39104640 | 32
(2 rows)
```

## STV\_LOCKS

Use the STV\_LOCKS table to view any current updates on tables in the database.

Amazon Redshift locks tables to prevent two users from updating the same table at the same time. While the STV\_LOCKS table shows all current table updates, query the [STL\\_TR\\_CONFLICT \(p. 1561\)](#) table to see a log of lock conflicts. Use the [SVV\\_TRANSACTIONS \(p. 1460\)](#) view to identify open transactions and lock contention issues.

STV\_LOCKS is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name         | Data type      | Description                                                 |
|---------------------|----------------|-------------------------------------------------------------|
| table_id            | bigint         | Table ID for the table acquiring the lock.                  |
| last_commit         | timestamp      | Timestamp for the last commit in the table.                 |
| last_update         | timestamp      | Timestamp for the last update for the table.                |
| lock_owner          | bigint         | Transaction ID associated with the lock.                    |
| lock_owner_pid      | bigint         | Process ID associated with the lock.                        |
| lock_owner_start_ts | timestamp      | Timestamp for the transaction start time.                   |
| lock_owner_end_ts   | timestamp      | Timestamp for the transaction end time.                     |
| lock_status         | character (22) | Status of the process either waiting for or holding a lock. |

### Sample query

To view all locks taking place in current transactions, type the following command:

```
select table_id, last_update, lock_owner, lock_owner_pid from stv_locks;
```

This query returns the following sample output, which displays three locks currently in effect:

```
table_id | last_update | lock_owner | lock_owner_pid
-----+-----+-----+-----+
100004 | 2008-12-23 10:08:48.882319 | 1043 | 5656
100003 | 2008-12-23 10:08:48.779543 | 1043 | 5656
100140 | 2008-12-23 10:08:48.021576 | 1043 | 5656
(3 rows)
```

## STV\_ML\_MODEL\_INFO

State information about the current state of the machine learning model.

STV\_ML\_MODEL\_INFO is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name    | Data type | Description                                                                                                                                                                                                                     |
|----------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| schema_name    | char(128) | The namespace of the model.                                                                                                                                                                                                     |
| user_name      | char(128) | The owner of the model.                                                                                                                                                                                                         |
| model_name     | char(128) | The name of the model.                                                                                                                                                                                                          |
| life_cycle     | char(20)  | The lifecycle status of the model.                                                                                                                                                                                              |
| is_refreshable | integer   | The state of the model whether it is refreshable if original tables and columns in the training query still exist and the user still has the permissions to them. Possible values are: 1 (refreshable) and 0 (not refreshable). |
| model_state    | char(128) | The current state of the model.                                                                                                                                                                                                 |

### Sample query

The following query displays the current state of machine learning models.

```
SELECT schema_name, model_name, model_state
FROM stv_ml_model_info;
-----+-----+-----+
schema_name | model_name | model_state
-----+-----+-----+
public | customer_churn_auto_model | Train Model On SageMaker In Progress
public | customer_churn_xgboost_model | Model is Ready
(2 row)
```

## STV\_MV\_DEPS

The STV\_MV\_DEPS table shows the dependencies of materialized views on other materialized views within Amazon Redshift.

For more information about materialized views, see [Creating materialized views in Amazon Redshift \(p. 198\)](#).

STV\_MV\_DEPS is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                 |
|-------------|-----------|-------------------------------------------------------------|
| db_name     | char(128) | The database that contains the specified materialized view. |

| Column name       | Data type | Description                                                                |
|-------------------|-----------|----------------------------------------------------------------------------|
| schema            | char(128) | The schema of the materialized view.                                       |
| name              | char(128) | The name of the materialized view.                                         |
| ref_schema        | char(128) | The materialized view schema on which this materialized view depends.      |
| ref_name          | char(128) | The name of the materialized view on which this materialized view depends. |
| ref_database_name | char(128) | The name of the database on which this materialized view depends.          |

## Sample query

The following query returns an output row that indicates that the materialized view `mv_over_foo` uses the materialized view `mv_foo` in its definition as a dependency.

```
CREATE SCHEMA test_ivm_setup;
CREATE TABLE test_ivm_setup.foo(a INT);
CREATE MATERIALIZED VIEW test_ivm_setup.mv_foo AS SELECT * FROM test_ivm_setup.foo;
CREATE MATERIALIZED VIEW test_ivm_setup.mv_over_foo AS SELECT * FROM test_ivm_setup.mv_foo;

SELECT * FROM stv_mv_deps;

db_name | schema | name | ref_schema | ref_name | ref_database_name
-----+-----+-----+-----+-----+-----+
dev | test_ivm_setup | mv_over_foo | test_ivm_setup | mv_foo | dev
```

## STV\_MV\_INFO

The STV\_MV\_INFO table contains a row for every materialized view, whether the data is stale, and state information.

For more information about materialized views, see [Creating materialized views in Amazon Redshift \(p. 198\)](#).

STV\_MV\_INFO is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name       | Data type | Description                                                                                                                                                                                                                      |
|-------------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| db_name           | char(128) | The database that contains the materialized view.                                                                                                                                                                                |
| schema            | char(128) | The schema of the database.                                                                                                                                                                                                      |
| name              | char(128) | The materialized view name.                                                                                                                                                                                                      |
| updated_up_to_xid | bigint    | Reserved for internal use.                                                                                                                                                                                                       |
| is_stale          | char(1)   | A <code>t</code> indicates that the materialized view is stale. A <code>stale</code> materialized view is one where the base tables have been updated but the materialized view hasn't been refreshed. The information might not |

| Column name                  | Data type              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------------------|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                              |                        | be accurate if a refresh hasn't been run since the last restart.<br><br>The <code>is_stale</code> column is always set to <code>t</code> if the materialized view depends on a mutable function. A mutable function returns a different result when given the same argument or arguments. For instance, most functions that return a date or timestamp are mutable functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <code>owner_user_name</code> | <code>char(128)</code> | The user who owns the materialized view.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <code>state</code>           | <code>integer</code>   | The state of the materialized view as follows: <ul style="list-style-type: none"> <li>• 0 – The materialized view is fully recomputed when refreshed.</li> <li>• 1 – The materialized view is incremental.</li> <li>• 101 – The materialized view can't be refreshed due to a dropped column. This constraint applies even if the column isn't used in the materialized view.</li> <li>• 102 – The materialized view can't be refreshed due to a changed column type. This constraint applies even if the column isn't used in the materialized view.</li> <li>• 103 – The materialized view can't be refreshed due to a renamed table.</li> <li>• 104 – The materialized view can't be refreshed due to a renamed column. This constraint applies even if the column isn't used in the materialized view.</li> <li>• 105 – The materialized view can't be refreshed due to a renamed schema.</li> </ul> |
| <code>autorewrite</code>     | <code>char(1)</code>   | A <code>t</code> indicates that the materialized view is eligible for automatic rewriting of queries.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <code>autorefresh</code>     | <code>char(1)</code>   | A <code>t</code> indicates that the materialized view can be automatically refreshed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

## Sample query

To view the state of all materialized views, run the following query.

```
select * from stv_mv_info;
```

This query returns the following sample output.

| db_name | schema         | name        | updated_upto_xid | is_stale | owner_user_name |
|---------|----------------|-------------|------------------|----------|-----------------|
| state   | autorefresh    | autorewrite |                  |          |                 |
| dev     | test_ivm_setup | mv          | 1031             | f        | catch-22        |
| 1       | 1              | 0           |                  |          |                 |
| dev     | test_ivm_setup | old_mv      | 988              | t        | lotr            |
| 1       | 0              | 1           |                  |          |                 |

## STV\_NODE\_STORAGE\_CAPACITY

The STV\_NODE\_STORAGE\_CAPACITY table shows details of total storage capacity and total used capacity for each node in a cluster. It contains a row for each node.

STV\_NODE\_STORAGE\_CAPACITY is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| node        | integer   | The node number.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| used        | integer   | The number of 1 MB disk blocks currently in use on the node. For RA3 node types, used blocks include both locally cached blocks and blocks persisted in Amazon S3.                                                                                                                                                                                                                                                                                                                                                                          |
| capacity    | integer   | The total storage capacity provisioned for the node in 1 MB blocks. The capacity includes space that is reserved by Amazon Redshift on DS2 or DC2 node types for internal use. The capacity is larger than the nominal node capacity, which is the amount of node space available for user data. For RA3 node types, this capacity is the same as the total managed storage quota for the cluster. For more information about capacity by node type, see <a href="#">Node type details</a> in the <i>Amazon Redshift Management Guide</i> . |

### Sample queries

#### Note

The results of the following examples vary based on the node specifications of your cluster. Add column capacity to your SQL SELECT to retrieve the capacity of your cluster.

The following query returns used space and total capacity in 1 MB disk blocks. This example ran on a two-node dc2.8xlarge cluster.

```
select node, used from stv_node_storage_capacity order by node;
```

This query returns the following sample output.

| node | used  |
|------|-------|
| 0    | 30597 |
| 1    | 27089 |

The following query returns used space and total capacity in 1 MB disk blocks. This example ran on a two-node ra3.16xlarge cluster.

```
select node, used from stv_node_storage_capacity order by node;
```

This query returns the following sample output.

| node | used  |
|------|-------|
| 0    | 30591 |
| 1    | 27103 |

## STV\_PARTITIONS

Use the STV\_PARTITIONS table to find out the disk speed performance and disk utilization for Amazon Redshift.

STV\_PARTITIONS contains one row per node per logical disk partition, or slice.

STV\_PARTITIONS is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name  | Data type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| owner        | integer   | Disk node that owns the partition.                                                                                                                                                                                                                                                                                                                                                                                                                          |
| host         | integer   | Node that is physically attached to the partition.                                                                                                                                                                                                                                                                                                                                                                                                          |
| diskno       | integer   | Disk containing the partition.                                                                                                                                                                                                                                                                                                                                                                                                                              |
| part_begin   | bigint    | Offset of the partition. Raw devices are logically partitioned to open space for mirror blocks.                                                                                                                                                                                                                                                                                                                                                             |
| part_end     | bigint    | End of the partition.                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| used         | integer   | Number of 1 MB disk blocks currently in use on the partition.                                                                                                                                                                                                                                                                                                                                                                                               |
| tossed       | integer   | Number of blocks that are ready to be deleted but are not yet removed because it is not safe to free their disk addresses. If the addresses were freed immediately, a pending transaction could write to the same location on disk. Therefore, these tossed blocks are released as of the next commit. Disk blocks might be marked as tossed, for example, when a table column is dropped, during INSERT operations, or during disk-based query operations. |
| capacity     | integer   | Total capacity of the partition in 1 MB disk blocks.                                                                                                                                                                                                                                                                                                                                                                                                        |
| reads        | bigint    | Number of reads that have occurred since the last cluster restart.                                                                                                                                                                                                                                                                                                                                                                                          |
| writes       | bigint    | Number of writes that have occurred since the last cluster restart.                                                                                                                                                                                                                                                                                                                                                                                         |
| seek_forward | integer   | Number of times that a request is not for the subsequent address given the previous request address.                                                                                                                                                                                                                                                                                                                                                        |
| seek_back    | integer   | Number of times that a request is not for the previous address given the subsequent address.                                                                                                                                                                                                                                                                                                                                                                |

| Column name | Data type      | Description                                                                                   |
|-------------|----------------|-----------------------------------------------------------------------------------------------|
| is_san      | integer        | Whether the partition belongs to a SAN. Valid values are <b>0</b> (false) or <b>1</b> (true). |
| failed      | integer        | This column is deprecated.                                                                    |
| mbps        | integer        | Disk speed in megabytes per second.                                                           |
| mount       | character(256) | Directory path to the device.                                                                 |

## Sample query

The following query returns the disk space used and capacity, in 1 MB disk blocks, and calculates disk utilization as a percentage of raw disk space. The raw disk space includes space that is reserved by Amazon Redshift for internal use, so it is larger than the nominal disk capacity, which is the amount of disk space available to the user. The **Percentage of Disk Space Used** metric on the **Performance** tab of the Amazon Redshift Management Console reports the percentage of nominal disk capacity used by your cluster. We recommend that you monitor the **Percentage of Disk Space Used** metric to maintain your usage within your cluster's nominal disk capacity.

**Important**

We strongly recommend that you do not exceed your cluster's nominal disk capacity. While it might be technically possible under certain circumstances, exceeding your nominal disk capacity decreases your cluster's fault tolerance and increases your risk of losing data.

This example was run on a two-node cluster with six logical disk partitions per node. Space is being used very evenly across the disks, with approximately 25% of each disk in use.

```
select owner, host, diskno, used, capacity,
(used-tossed)/capacity::numeric *100 as pctused
from stv_partitions order by owner;

owner | host | diskno | used | capacity | pctused
-----+-----+-----+-----+-----+
 0 | 0 | 0 | 236480 | 949954 | 24.9
 0 | 0 | 1 | 236420 | 949954 | 24.9
 0 | 0 | 2 | 236440 | 949954 | 24.9
 0 | 1 | 2 | 235150 | 949954 | 24.8
 0 | 1 | 1 | 237100 | 949954 | 25.0
 0 | 1 | 0 | 237090 | 949954 | 25.0
 1 | 1 | 0 | 236310 | 949954 | 24.9
 1 | 1 | 1 | 236300 | 949954 | 24.9
 1 | 1 | 2 | 236320 | 949954 | 24.9
 1 | 0 | 2 | 237910 | 949954 | 25.0
 1 | 0 | 1 | 235640 | 949954 | 24.8
 1 | 0 | 0 | 235380 | 949954 | 24.8

(12 rows)
```

## STV\_QUERY\_METRICS

Contains metrics information, such as the number of rows processed, CPU usage, input/output, and disk use, for active queries running in user-defined query queues (service classes). To view metrics for queries that have completed, see the [STL\\_QUERY\\_METRICS \(p. 1542\)](#) system table.

Query metrics are sampled at one second intervals. As a result, different runs of the same query might return slightly different times. Also, query segments that run in less than 1 second might not be recorded.

`STV_QUERY_METRICS` tracks and aggregates metrics at the query, segment, and step level. For information about query segments and steps, see [Query planning and execution workflow \(p. 463\)](#). Many metrics (such as `max_rows`, `cpu_time`, and so on) are summed across node slices. For more information about node slices, see [Data warehouse system architecture \(p. 3\)](#).

To determine the level at which the row reports metrics, examine the `segment` and `step_type` columns:

- If both `segment` and `step_type` are -1, then the row reports metrics at the query level.
- If `segment` is not -1 and `step_type` is -1, then the row reports metrics at the segment level.
- If both `segment` and `step_type` are not -1, then the row reports metrics at the step level.

This table is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name                | Data type | Description                                                                                                                                                                                                                                                                         |
|----------------------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>userid</code>        | integer   | ID of the user that ran the query that generated the entry.                                                                                                                                                                                                                         |
| <code>service_class</code> | integer   | ID for the WLM query queue (service class). Query queues are defined in the WLM configuration. Metrics are reported only for user-defined queues.                                                                                                                                   |
| <code>query</code>         | integer   | Query ID. The <code>query</code> column can be used to join other system tables and views.                                                                                                                                                                                          |
| <code>starttime</code>     | timestamp | Time in UTC that the query started executing, with 6 digits of precision for fractional seconds. For example: 2009-06-12 11:29:19.131358.                                                                                                                                           |
| <code>slices</code>        | integer   | Number of slices for the cluster.                                                                                                                                                                                                                                                   |
| <code>segment</code>       | integer   | Segment number. A query consists of multiple segments, and each segment consists of one or more steps. Query segments can run in parallel. Each segment runs in a single process. If the <code>segment</code> value is -1, metrics segment values are rolled up to the query level. |
| <code>step_type</code>     | integer   | Type of step that ran. For a description of step types, see <a href="#">Step types (p. 1594)</a> .                                                                                                                                                                                  |
| <code>rows</code>          | bigint    | Number of rows processed by a step.                                                                                                                                                                                                                                                 |
| <code>max_rows</code>      | bigint    | Maximum number of rows output for a step, aggregated across all slices.                                                                                                                                                                                                             |
| <code>cpu_time</code>      | bigint    | CPU time used, in microseconds. At the segment level, the total CPU time for the segment across all slices. At the query level, the sum of CPU time for the query across all slices and segments.                                                                                   |
| <code>max_cpu_time</code>  | bigint    | Maximum CPU time used, in microseconds. At the segment level, the maximum CPU time used by the segment across all slices. At the query level, the maximum CPU time used by any query segment.                                                                                       |
| <code>blocks_read</code>   | bigint    | Number of 1 MB blocks read by the query or segment.                                                                                                                                                                                                                                 |

| Column name         | Data type | Description                                                                                                                                                                                                                                                                                                                        |
|---------------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| max_blocks_read     | bigint    | Maximum number of 1 MB blocks read by the segment, aggregated across all slices. At the segment level, the maximum number of 1 MB blocks read for the segment across all slices. At the query level, the maximum number of 1 MB blocks read by any query segment.                                                                  |
| run_time            | bigint    | Total run time, summed across slices. Run time doesn't include wait time.<br><br>At the segment level, the run time for the segment, summed across all slices. At the query level, the run time for the query summed across all slices and segments. Because this value is a sum, run time is not related to query execution time. |
| max_run_time        | bigint    | The maximum elapsed time for a segment, in microseconds. At the segment level, the maximum run time for the segment across all slices. At the query level, the maximum run time for any query segment.                                                                                                                             |
| max_blocks_to_disk  | bigint    | The maximum amount of disk space used to write intermediate results, in 1 MB blocks. At the segment level, the maximum amount of disk space used by the segment across all slices. At the query level, the maximum amount of disk space used by any query segment.                                                                 |
| blocks_to_disk      | bigint    | The amount of disk space used by a query or segment to write intermediate results, in 1 MB blocks.                                                                                                                                                                                                                                 |
| step                | integer   | Query step that ran.                                                                                                                                                                                                                                                                                                               |
| max_query_scan_size | bigint    | The maximum size of data scanned by a query, in MB. At the segment level, the maximum size of data scanned by the segment across all slices. At the query level, the maximum size of data scanned by any query segment.                                                                                                            |
| query_scan_size     | bigint    | The size of data scanned by a query, in MB.                                                                                                                                                                                                                                                                                        |
| query_priority      | integer   | The priority of the query. Possible values are -1, 0, 1, 2, 3, and 4, where -1 means that query priority isn't supported.                                                                                                                                                                                                          |
| query_queue_time    | bigint    | The amount of time in microseconds that the query was queued.                                                                                                                                                                                                                                                                      |

## Step types

The following table lists step types relevant to database users. The table doesn't list step types that are for internal use only. If step type is -1, the metric is not reported at the step level.

| Step type | Description    |
|-----------|----------------|
| 1         | Scan table     |
| 2         | Insert rows    |
| 3         | Aggregate rows |
| 6         | Sort step      |

| Step type | Description                                           |
|-----------|-------------------------------------------------------|
| 7         | Merge step                                            |
| 8         | Distribution step                                     |
| 9         | Broadcast distribution step                           |
| 10        | Hash join                                             |
| 11        | Merge join                                            |
| 12        | Save step                                             |
| 14        | Hash                                                  |
| 15        | Nested loop join                                      |
| 16        | Project fields and expressions                        |
| 17        | Limit the number of rows returned                     |
| 18        | Unique                                                |
| 20        | Delete rows                                           |
| 26        | Limit the number of sorted rows returned              |
| 29        | Compute a window function                             |
| 32        | UDF                                                   |
| 33        | Unique                                                |
| 37        | Return rows from the leader node to the client        |
| 38        | Return rows from the compute nodes to the leader node |
| 40        | Spectrum scan.                                        |

## Sample query

To find active queries with high CPU time (more than 1,000 seconds), run the following query.

```
select query, cpu_time / 1000000 as cpu_seconds
from stv_query_metrics where segment = -1 and cpu_time > 1000000000
order by cpu_time;

query | cpu_seconds
-----+-----
25775 | 9540
```

To find active queries with a nested loop join that returned more than one million rows, run the following query.

```
select query, rows
from stv_query_metrics
where step_type = 15 and rows > 1000000
order by rows;

query | rows
```

|                    |
|--------------------|
| -----+-----        |
| 25775   1580225854 |

To find active queries that have run for more than 60 seconds and have used less than 10 seconds of CPU time, run the following query.

```
select query, run_time/1000000 as run_time_seconds
from stv_query_metrics
where segment = -1 and run_time > 6000000 and cpu_time < 10000000;

query | run_time_seconds
-----+-----
25775 | 114
```

## STV\_RECENTS

Use the STV\_RECENTS table to find out information about the currently active and recently run queries against a database.

STV\_RECENTS is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type      | Description                                                                                               |
|-------------|----------------|-----------------------------------------------------------------------------------------------------------|
| userid      | integer        | ID of user who generated entry.                                                                           |
| status      | character(20)  | Query status. Valid values are <b>Running</b> , <b>Done</b> .                                             |
| starttime   | timestamp      | Time that the query started.                                                                              |
| duration    | integer        | Number of microseconds since the session started.                                                         |
| user_name   | character(50)  | User name who ran the process.                                                                            |
| db_name     | character(50)  | Name of the database.                                                                                     |
| query       | character(600) | Query text, up to 600 characters. Any additional characters are truncated.                                |
| pid         | integer        | Process ID for the session associated with the query, which is always -1 for queries that have completed. |

## Sample queries

To determine what queries are currently running against the database, type the following query:

```
select user_name, db_name, pid, query
from stv_recents
where status = 'Running';
```

The sample output below shows a single query running on the TICKIT database:

|                                   |
|-----------------------------------|
| user_name   db_name   pid   query |
|-----------------------------------|

```
dwuser | tickit | 19996 |select venuename, venueseats from
venue where venueseats > 50000 order by venueseats desc;
```

The following example returns a list of queries (if any) that are running or waiting in queue to run:

```
select * from stv_recents where status<>'Done';

status | starttime | duration |user_name|db_name| query | pid
-----+-----+-----+-----+-----+-----+
Running| 2010-04-21 16:11... | 281566454| dwuser |tickit | select ...| 23347
```

This query does not return results unless you are running a number of concurrent queries and some of those queries are in queue.

The following example extends the previous example. In this case, queries that are truly "in flight" (running, not waiting) are excluded from the result:

```
select * from stv_recents where status<>'Done'
and pid not in (select pid from stv_inflight);
...
```

## STV\_SESSIONS

Use the STV\_SESSIONS table to view information about the active user sessions for Amazon Redshift.

To view the session history, use the [STL\\_SESSIONS \(p. 1557\)](#) table, rather than STV\_SESSIONS.

All rows in STV\_SESSIONS, including rows generated by other users, are visible to all users.

### Table columns

| Column name | Data type     | Description                                                                                                                |
|-------------|---------------|----------------------------------------------------------------------------------------------------------------------------|
| starttime   | timestamp     | Time that the session started.                                                                                             |
| process     | integer       | Process ID for the session.                                                                                                |
| user_name   | character(50) | User associated with the session.                                                                                          |
| db_name     | character(50) | Name of the database associated with the session.                                                                          |
| timeout_sec | int           | The maximum time in seconds that a session remains inactive or idle before timing out. 0 indicates that no timeout is set. |

### Sample queries

To perform a quick check to see if any other users are currently logged into Amazon Redshift, type the following query:

```
select count(*)
from stv_sessions;
```

If the result is greater than one, then at least one other user is currently logged in to the database.

To view all active sessions for Amazon Redshift, type the following query:

```
select *
from stv_sessions;
```

The following result shows four active sessions running on Amazon Redshift:

| starttime<br>  timeout_sec      | process | user_name                | db_name |
|---------------------------------|---------|--------------------------|---------|
| 2018-08-06 08:44:07.50<br>  0   | 13779   | IAMA:aws_admin:admin_grp | dev     |
| 2008-08-06 08:54:20.50<br>  120 | 19829   | dwuser                   | dev     |
| 2008-08-06 08:56:34.50<br>  120 | 20279   | dwuser                   | dev     |
| 2008-08-06 08:55:00.50<br>  0   | 19996   | dwuser                   | ticket  |
| (3 rows)                        |         |                          |         |

The user name prefixed with IAMA indicates that the user signed on using federated single sign-on. For more information, see [Using IAM authentication to generate database user credentials](#).

## STV\_SLICES

Use the STV\_SLICES table to view the current mapping of a slice to a node.

The information in STV\_SLICES is used mainly for investigation purposes.

STV\_SLICES is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type    | Description                                |
|-------------|--------------|--------------------------------------------|
| node        | integer      | Cluster node where the slice is located.   |
| slice       | integer      | Node slice.                                |
| localslice  | integer      | This information is for internal use only. |
| type        | character(1) | This information is for internal use only. |

### Sample query

To view which cluster nodes are managing which slices, type the following query:

```
select node, slice from stv_slices;
```

This query returns the following sample output:

| node | slice |
|------|-------|
| 0    | 2     |
| 0    | 3     |
| 0    | 1     |

|          |  |   |
|----------|--|---|
| 0        |  | 0 |
| (4 rows) |  |   |

## STV\_STARTUP\_RECOVERY\_STATE

Records the state of tables that are temporarily locked during cluster restart operations. Amazon Redshift places a temporary lock on tables while they are being processed to resolve stale transactions following a cluster restart.

STV\_STARTUP\_RECOVERY\_STATE is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type      | Description  |
|-------------|----------------|--------------|
| db_id       | integer        | Database ID. |
| table_id    | integer        | Table ID.    |
| table_name  | character(137) | Table name.  |

### Sample queries

To monitor which tables are temporarily locked, run the following query after a cluster restart.

```
select * from STV_STARTUP_RECOVERY_STATE;

 db_id | tbl_id | table_name
-----+-----+-----
 100044 | 100058 | lineorder
 100044 | 100068 | part
 100044 | 100072 | customer
 100044 | 100192 | supplier
(4 rows)
```

## STV\_TBL\_PERM

The STV\_TBL\_PERM table contains information about the permanent tables in Amazon Redshift, including temporary tables created by a user for the current session. STV\_TBL\_PERM contains information for all tables in all databases.

This table differs from [STV\\_TBL\\_TRANS \(p. 1601\)](#), which contains information about transient database tables that the system creates during query processing.

STV\_TBL\_PERM is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                        |
|-------------|-----------|------------------------------------|
| slice       | integer   | Node slice allocated to the table. |
| id          | integer   | Table ID.                          |

| Column name     | Data type     | Description                                                                                                                                                                                                                                   |
|-----------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| name            | character(72) | Table name.                                                                                                                                                                                                                                   |
| rows            | bigint        | Number of data rows in the slice.                                                                                                                                                                                                             |
| sorted_rows     | bigint        | Number of rows in the slice that are already sorted on disk. If this number does not match the ROWS number, vacuum the table to resort the rows.                                                                                              |
| temp            | integer       | Whether or not the table is a temporary table. 0 = false; 1 = true.                                                                                                                                                                           |
| db_id           | integer       | ID of the database where the table was created.                                                                                                                                                                                               |
| insert_pristine | integer       | For internal use.                                                                                                                                                                                                                             |
| delete_pristine | integer       | For internal use.                                                                                                                                                                                                                             |
| backup          | integer       | Value that indicates whether the table is included in cluster snapshots. 0 = no; 1 = yes. For more information, see the <a href="#">BACKUP (p. 829)</a> parameter for the CREATE TABLE command.                                               |
| dist_style      | integer       | Distribution style of the table that the slice belongs to. For information on the values, see <a href="#">Viewing distribution styles (p. 52)</a> . For information on distribution styles, see <a href="#">Distribution styles (p. 51)</a> . |
| block_count     | integer       | Number of blocks used by the slice. The value is -1 when the block count can't be calculated.                                                                                                                                                 |

## Sample queries

The following query returns a list of distinct table IDs and names:

```
select distinct id, name
from stv_tbl_perm order by name;

id | name
---+-----
100571 | category
100575 | date
100580 | event
100596 | listing
100003 | padb_config_harvest
100612 | sales
...
```

Other system tables use table IDs, so knowing which table ID corresponds to a certain table can be very useful. In this example, SELECT DISTINCT is used to remove the duplicates (tables are distributed across multiple slices).

To determine the number of blocks used by each column in the VENUE table, type the following query:

```
select col, count(*)
from stv_blocklist, stv_tbl_perm
where stv_blocklist.tbl = stv_tbl_perm.id
and stv_blocklist.slice = stv_tbl_perm.slice
and stv_tbl_perm.name = 'venue'
group by col
order by col;
```

| col      | count |
|----------|-------|
| 0        | 8     |
| 1        | 8     |
| 2        | 8     |
| 3        | 8     |
| 4        | 8     |
| 5        | 8     |
| 6        | 8     |
| 7        | 8     |
| (8 rows) |       |

## Usage notes

The ROWS column includes counts of deleted rows that have not been vacuumed (or have been vacuumed but with the SORT ONLY option). Therefore, the SUM of the ROWS column in the STV\_TBL\_PERM table might not match the COUNT(\*) result when you query a given table directly. For example, if 2 rows are deleted from VENUE, the COUNT(\*) result is 200 but the SUM(ROWS) result is still 202:

```
delete from venue
where venueid in (1,2);

select count(*) from venue;
count

200
(1 row)

select trim(name) tablename, sum(rows)
from stv_tbl_perm where name='venue' group by name;

tablename | sum
-----+-----
venue | 202
(1 row)
```

To synchronize the data in STV\_TBL\_PERM, run a full vacuum the VENUE table.

```
vacuum venue;

select trim(name) tablename, sum(rows)
from stv_tbl_perm
where name='venue'
group by name;

tablename | sum
-----+-----
venue | 200
(1 row)
```

## STV\_TBL\_TRANS

Use the STV\_TBL\_TRANS table to find out information about the transient database tables that are currently in memory.

Transient tables are typically temporary row sets that are used as intermediate results while a query runs. STV\_TBL\_TRANS differs from [STV\\_TBL\\_PERM \(p. 1599\)](#) in that STV\_TBL\_PERM contains information about permanent database tables.

STV\_TBL\_TRANS is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name    | Data type | Description                                                                                                                              |
|----------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------|
| slice          | integer   | Node slice allocated to the table.                                                                                                       |
| id             | integer   | Table ID.                                                                                                                                |
| rows           | bigint    | Number of data rows in the table.                                                                                                        |
| size           | bigint    | Number of bytes allocated to the table.                                                                                                  |
| query_id       | bigint    | Query ID.                                                                                                                                |
| ref_cnt        | integer   | Number of references.                                                                                                                    |
| from_suspended | integer   | Whether or not this table was created during a query that is now suspended.                                                              |
| prep_swap      | integer   | Whether or not this transient table is prepared to swap to disk if needed. (The swap will only occur in situations where memory is low.) |

## Sample queries

To view transient table information for a query with a query ID of 90, type the following command:

```
select slice, id, rows, size, query_id, ref_cnt
from stv_tbl_trans
where query_id = 90;
```

This query returns the transient table information for query 90, as shown in the following sample output:

| slice | id | rows | size | query_id | ref_cnt | from_suspended | prep_swap |
|-------|----|------|------|----------|---------|----------------|-----------|
| 1013  | 95 | 0    | 0    | 90       | 4       | 0              | 0         |
| 7     | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 10    | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 17    | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 14    | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 3     | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 1013  | 99 | 0    | 0    | 90       | 4       | 0              | 0         |
| 9     | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 5     | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 19    | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 2     | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 1013  | 98 | 0    | 0    | 90       | 4       | 0              | 0         |
| 13    | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 1     | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 1013  | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 6     | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 11    | 96 | 0    | 0    | 90       | 4       | 0              | 0         |
| 15    | 96 | 0    | 0    | 90       | 4       | 0              | 0         |

|    |  |    |  |   |  |   |  |    |  |   |  |   |  |   |
|----|--|----|--|---|--|---|--|----|--|---|--|---|--|---|
| 18 |  | 96 |  | 0 |  | 0 |  | 90 |  | 4 |  | 0 |  | 0 |
|----|--|----|--|---|--|---|--|----|--|---|--|---|--|---|

In this example, you can see that the query data involves tables 95, 96, and 98. Because zero bytes are allocated to this table, this query can run in memory.

## STV\_WLM\_QMR\_CONFIG

Records the configuration for WLM query monitoring rules (QMR). For more information, see [WLM query monitoring rules \(p. 527\)](#).

STV\_WLM\_QMR\_CONFIG is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name     | Data type      | Description                                                                                                                                                                                                                             |
|-----------------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| service_class   | integer        | ID for the WLM query queue (service class). Query queues are defined in the WLM configuration. Rules can be defined only for user-defined queues. For a list of service class IDs, see <a href="#">WLM service class IDs (p. 534)</a> . |
| rule_name       | character(256) | Name of the query monitoring rule.                                                                                                                                                                                                      |
| action          | character(256) | Rule action. Possible values are log, hop, abort, and change_query_priority.                                                                                                                                                            |
| metric_name     | character(256) | Name of the metric.                                                                                                                                                                                                                     |
| metric_operator | character(256) | The metric operator. Possible values are >, =, <.                                                                                                                                                                                       |
| metric_value    | double         | The threshold value for the specified metric that triggers an action.                                                                                                                                                                   |
| action_value    | character(256) | If action is change_query_priority, then possible values are highest, high, normal, low, and lowest.<br><br>If action is log, hop, or abort then the value is empty.                                                                    |

### Sample query

To view the QMR rule definitions for all service classes greater than 5 (which includes user-defined queues), run the following query. For a list of service class IDs, see [WLM service class IDs \(p. 534\)](#).

```
Select *
from stv_wlm_qmr_config
where service_class > 5
order by service_class;
```

## STV\_WLM\_CLASSIFICATION\_CONFIG

Contains the current classification rules for WLM.

STV\_WLM\_CLASSIFICATION\_CONFIG is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name                 | Data type      | Description                                                                                             |
|-----------------------------|----------------|---------------------------------------------------------------------------------------------------------|
| id                          | integer        | Service class ID. For a list of service class IDs, see <a href="#">WLM service class IDs (p. 534)</a> . |
| condition                   | character(128) | Query conditions.                                                                                       |
| action_seq                  | integer        | Reserved for system use.                                                                                |
| action                      | character(64)  | Reserved for system use.                                                                                |
| action_service_classinteger |                | The service class where the action takes place.                                                         |

## Sample query

```
select * from STV_WLM_CLASSIFICATION_CONFIG;

+-----+-----+-----+
| id | condition | action_seq | action |
|-----+-----+-----+
1	(system user) and (query group: health)	0	assign
1		0	assign
2	(system user) and (query group: metrics)	0	assign
2		0	assign
3	(system user) and (query group: cmstats)	0	assign
3		0	assign
4	(system user)	0	assign
4		0	assign
5	(super user) and (query group: superuser)	0	assign
5		0	assign
6	(query group: querygroup1)	0	assign
6		0	assign
7	(user group: usergroup1)	0	assign
7		0	assign
8	(user group: usergroup2)	0	assign
8		0	assign
9	(query group: querygroup3)	0	assign
9		0	assign
10	(query group: querygroup4)	0	assign
10		0	assign
11	(user group: usergroup4)	0	assign
11		0	assign
12	(query group: querygroup*)	0	assign
12		0	assign
13	(user group: usergroup*)	0	assign
13		0	assign
14	(querytype: any)	0	assign
14		0	assign
+-----+-----+-----+
(4 rows)
```

## STV\_WLM\_QUERY\_QUEUE\_STATE

Records the current state of the query queues for the service classes.

STV\_WLM\_QUERY\_QUEUE\_STATE is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name   | Data type | Description                                                                                                                                                                            |
|---------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| service_class | integer   | ID for the service class. For a list of service class IDs, see <a href="#">WLM service class IDs (p. 534)</a> .                                                                        |
| position      | integer   | Position of the query in the queue. The query with the smallest <b>position</b> value runs next.                                                                                       |
| task          | integer   | ID used to track a query through the workload manager. Can be associated with multiple query IDs. If a query is restarted, the query is assigned a new query ID but not a new task ID. |
| query         | integer   | Query ID. If a query is restarted, the query is assigned a new query ID but not a new task ID.                                                                                         |
| slot_count    | integer   | Number of WLM query slots.                                                                                                                                                             |
| start_time    | timestamp | Time that the query entered the queue.                                                                                                                                                 |
| queue_time    | bigint    | Number of microseconds that the query has been in the queue.                                                                                                                           |

## Sample query

The following query shows the queries in the queue for service classes greater than 4.

```
select * from stv_wlm_query_queue_state
where service_class > 4
order by service_class;
```

This query returns the following sample output.

| service_class | position | task | query | slot_count | start_time | queue_time                 |
|---------------|----------|------|-------|------------|------------|----------------------------|
| 20937257      | 5        | 0    | 455   | 476        | 5          | 2010-10-06 13:18:24.065838 |
| 18350191      | 6        | 1    | 456   | 478        | 5          | 2010-10-06 13:18:26.652906 |

(2 rows)

## STV\_WLM\_QUERY\_STATE

Records the current state of queries being tracked by WLM.

STV\_WLM\_QUERY\_STATE is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name    | Data type     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| xid            | integer       | Transaction ID of the query or subquery.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| task           | integer       | ID used to track a query through the workload manager. Can be associated with multiple query IDs. If a query is restarted, the query is assigned a new query ID but not a new task ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| query          | integer       | Query ID. If a query is restarted, the query is assigned a new query ID but not a new task ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| service_class  | integer       | ID for the service class. For a list of service class IDs, see <a href="#">WLM service class IDs (p. 534)</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| slot_count     | integer       | Number of WLM query slots.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| wlm_start_time | timestamp     | Time that the query entered the system table queue or short query queue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| state          | character(16) | <p>Current state of the query or subquery.</p> <p>Possible values are the following:</p> <ul style="list-style-type: none"> <li>• Classified – Query has been assigned to a service class.</li> <li>• Completed – Query is finished running. The query either ran successfully or was canceled. For the final state, check the results of <a href="#">STL_QUERY (p. 1539)</a>.</li> <li>• Dequeued – Internal use only.</li> <li>• Evicted – Query has been evicted from the service class for restart.</li> <li>• Evicting – Query is being evicted from the service class for restart.</li> <li>• Initialized – Internal use only.</li> <li>• Invalid – Internal use only.</li> <li>• Queued – Query was sent to the query queue because no slots were available to run it.</li> <li>• QueuedWaiting – Query is waiting in the query queue.</li> <li>• Rejected – Internal use only.</li> <li>• Returning – Query is returning results to the client.</li> <li>• Running – Query is running.</li> <li>• TaskAssigned – Internal use only.</li> </ul> |
| queue_time     | bigint        | Number of microseconds that the query has spent in the queue.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| exec_time      | bigint        | Number of microseconds that the query has been running.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| query_priority | char(20)      | The priority of the query. Possible values are n/a, lowest, low, normal, high, and highest, where n/a means that query priority isn't supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

## Sample query

The following query displays all currently executing queries in service classes greater than 4. For a list of service class IDs, see [WLM service class IDs \(p. 534\)](#).

```
select xid, query, trim(state) as state, queue_time, exec_time
from stv_wlm_query_state
where service_class > 4;
```

This query returns the following sample output:

| xid    | query | state   | queue_time | exec_time  |
|--------|-------|---------|------------|------------|
| 100813 | 25942 | Running | 0          | 1369029    |
| 100074 | 25775 | Running | 0          | 2221589242 |

## STV\_WLM\_QUERY\_TASK\_STATE

Contains the current state of service class query tasks.

STV\_WLM\_QUERY\_TASK\_STATE is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name   | Data type | Description                                                                                                                                                                            |
|---------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| service_class | integer   | ID for the service class. For a list of service class IDs, see <a href="#">WLM service class IDs (p. 534)</a> .                                                                        |
| task          | integer   | ID used to track a query through the workload manager. Can be associated with multiple query IDs. If a query is restarted, the query is assigned a new query ID but not a new task ID. |
| query         | integer   | Query ID. If a query is restarted, the query is assigned a new query ID but not a new task ID.                                                                                         |
| slot_count    | integer   | Number of WLM query slots.                                                                                                                                                             |
| start_time    | timestamp | Time that the query began executing.                                                                                                                                                   |
| exec_time     | bigint    | Number of microseconds that the query has been executing.                                                                                                                              |

## Sample query

The following query displays the current state of queries in service classes greater than 4. For a list of service class IDs, see [WLM service class IDs \(p. 534\)](#).

```
select * from stv_wlm_query_task_state
where service_class > 4;
```

This query returns the following sample output:

| service_class | task | query | start_time                 | exec_time |
|---------------|------|-------|----------------------------|-----------|
| 5             | 466  | 491   | 2010-10-06 13:29:23.063787 | 357618748 |
| (1 row)       |      |       |                            |           |

## STV\_WLM\_SERVICE\_CLASS\_CONFIG

Records the service class configurations for WLM.

STV\_WLM\_SERVICE\_CLASS\_CONFIG is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name              | Data type     | Description                                                                                                                                                                                                                                                            |
|--------------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| service_class            | integer       | ID for the service class. For a list of service class IDs, see <a href="#">WLM service class IDs (p. 534)</a> .                                                                                                                                                        |
| queueing_strategy        | character(32) | Reserved for system use.                                                                                                                                                                                                                                               |
| num_query_tasks          | integer       | Current actual concurrency level of the service class. If num_query_tasks and target_num_query_tasks are different, a dynamic WLM transition is in process. A value of -1 indicates that <b>Auto WLM</b> is configured.                                                |
| target_num_query_tasks   | integer       | Concurrency level set by the most recent WLM configuration change.                                                                                                                                                                                                     |
| evictable                | character(8)  | Reserved for system use.                                                                                                                                                                                                                                               |
| eviction_threshold       | bigint        | Reserved for system use.                                                                                                                                                                                                                                               |
| query_working_mem        | integer       | Current actual amount of working memory, in MB per slot, per node, assigned to the service class. If query_working_mem and target_query_working_mem are different, a dynamic WLM transition is in process. A value of -1 indicates than <b>Auto WLM</b> is configured. |
| target_query_working_mem | integer       | The amount of working memory, in MB per slot, per node, set by the most recent WLM configuration change.                                                                                                                                                               |
| min_step_mem             | integer       | Reserved for system use.                                                                                                                                                                                                                                               |
| name                     | character(64) | The name of the service class.                                                                                                                                                                                                                                         |
| max_execution_time       | bigint        | Number of milliseconds that the query can run before being terminated.                                                                                                                                                                                                 |
| user_group_wild_card     | Boolean       | If TRUE, the WLM queue treats an asterisk (*) as a wildcard character in user group strings in the WLM configuration.                                                                                                                                                  |
| query_group_wild_card    | Boolean       | If TRUE, the WLM queue treats an asterisk (*) as a wildcard character in query group strings in the WLM configuration.                                                                                                                                                 |
| concurrency_scaling      | character(20) | Describes if the concurrency scaling is on or off.                                                                                                                                                                                                                     |
| query_priority           | character(20) | The value of the query priority.                                                                                                                                                                                                                                       |

## Sample query

The first user-defined service class is service class 6, which is named Service class #1. The following query displays the current configuration for service classes greater than 4. For a list of service class IDs, see [WLM service class IDs \(p. 534\)](#).

```
select rtrim(name) as name,
num_query_tasks as slots,
query_working_mem as mem,
max_execution_time as max_time,
user_group_wild_card as user_wildcard,
query_group_wild_card as query_wildcard
from stv_wlm_service_class_config
where service_class > 4;

name | slots | mem | max_time | user_wildcard | query_wildcard
-----+-----+-----+-----+-----+-----+
Service class for super user | 1 | 535 | 0 | false | false
Queue 1 | 5 | 125 | 0 | false | false
Queue 2 | 5 | 125 | 0 | false | false
Queue 3 | 5 | 125 | 0 | false | false
Queue 4 | 5 | 627 | 0 | false | false
Queue 5 | 5 | 125 | 0 | true | true
Default queue | 5 | 125 | 0 | false | false
```

The following query shows the status of a dynamic WLM transition. While the transition is in process, num\_query\_tasks and target\_query\_working\_mem are updated until they equal the target values. For more information, see [WLM dynamic and static configuration properties \(p. 524\)](#).

```
select rtrim(name) as name,
num_query_tasks as slots,
target_num_query_tasks as target_slots,
query_working_mem as memory,
target_query_working_mem as target_memory
from stv_wlm_service_class_config
where num_query_tasks > target_num_query_tasks
or query_working_mem > target_query_working_mem
and service_class > 5;

name | slots | target_slots | memory | target_mem
-----+-----+-----+-----+-----+
Queue 3 | 5 | 15 | 125 | 375
Queue 5 | 10 | 5 | 250 | 125
(2 rows)
```

## STV\_WLM\_SERVICE\_CLASS\_STATE

Contains the current state of the service classes.

STV\_WLM\_SERVICE\_CLASS\_STATE is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name   | Data type | Description                                                                                                     |
|---------------|-----------|-----------------------------------------------------------------------------------------------------------------|
| service_class | integer   | ID for the service class. For a list of service class IDs, see <a href="#">WLM service class IDs (p. 534)</a> . |

| Column name                      | Data type | Description                                                                                                                                                                                                           |
|----------------------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| num_queued_queries               | integer   | Number of queries currently in the queue.                                                                                                                                                                             |
| num_executing_queries            | integer   | Number of queries currently executing.                                                                                                                                                                                |
| num_serviced_queries             | integer   | Number of queries that have ever been in the service class.                                                                                                                                                           |
| num_executed_queries             | integer   | Number of queries that have run since Amazon Redshift was restarted.                                                                                                                                                  |
| num_evicted_queries              | integer   | Number of queries that have been evicted since Amazon Redshift was restarted. Some of the reasons for an evicted query include a WLM timeout, a QMR hop action, and a query failing on a concurrency scaling cluster. |
| num_concurrency_scaling_integers | integer   | Number of queries run on a concurrency scaling cluster since Amazon Redshift was restarted.                                                                                                                           |

## Sample query

The following query displays the state for service classes greater than 5. For a list of service class IDs, see [WLM service class IDs \(p. 534\)](#).

```
select service_class, num_executing_queries,
 num_executed_queries
 from stv_wlm_service_class_state
 where service_class > 5
 order by service_class;
```

| service_class | num_executing_queries | num_executed_queries |
|---------------|-----------------------|----------------------|
| 6             | 1                     | 222                  |
| 7             | 0                     | 135                  |
| 8             | 1                     | 39                   |

(3 rows)

## SVCS views for main and concurrency scaling clusters

SVCS system views with the prefix SVCS provide details about queries on both the main and concurrency scaling clusters. The views are similar to the tables with the prefix STL except that the STL tables provide information only for queries run on the main cluster.

### Topics

- [SVCS\\_ALERT\\_EVENT\\_LOG \(p. 1611\)](#)
- [SVCS\\_COMPILE \(p. 1612\)](#)
- [SVCS\\_CONCURRENCY\\_SCALING\\_USAGE \(p. 1614\)](#)
- [SVCS\\_EXPLAIN \(p. 1614\)](#)
- [SVCS\\_PLAN\\_INFO \(p. 1616\)](#)
- [SVCS\\_QUERY\\_SUMMARY \(p. 1618\)](#)
- [SVCS\\_S3LIST \(p. 1621\)](#)
- [SVCS\\_S3LOG \(p. 1622\)](#)
- [SVCS\\_S3PARTITION\\_SUMMARY \(p. 1622\)](#)

- [SVCS\\_S3QUERY\\_SUMMARY \(p. 1623\)](#)
- [SVCS\\_STREAM\\_SEGS \(p. 1626\)](#)
- [SVCS\\_UNLOAD\\_LOG \(p. 1627\)](#)

## SVCS\_ALERT\_EVENT\_LOG

Records an alert when the query optimizer identifies conditions that might indicate performance issues. This view is derived from the STL\_ALERT\_EVENT\_LOG system table but doesn't show slice-level for queries run on a concurrency scaling cluster. Use the SVCS\_ALERT\_EVENT\_LOG table to identify opportunities to improve query performance.

A query consists of multiple segments, and each segment consists of one or more steps. For more information, see [Query processing \(p. 463\)](#).

**Note**

System views with the prefix SVCS provide details about queries on both the main and concurrency scaling clusters. The views are similar to the tables with the prefix STL except that the STL tables provide information only for queries run on the main cluster.

SVCS\_ALERT\_EVENT\_LOG is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type       | Description                                                                                                                                                                        |
|-------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer         | ID of the user who generated the entry.                                                                                                                                            |
| query       | integer         | Query ID. The query column can be used to join other system tables and views.                                                                                                      |
| segment     | integer         | Number that identifies the query segment.                                                                                                                                          |
| step        | integer         | Query step that ran.                                                                                                                                                               |
| pid         | integer         | Process ID associated with the statement and slice. The same query might have multiple PIDs if it runs on multiple slices.                                                         |
| xid         | bigint          | Transaction ID associated with the statement.                                                                                                                                      |
| event       | character(1024) | Description of the alert event.                                                                                                                                                    |
| solution    | character(1024) | Recommended solution.                                                                                                                                                              |
| event_time  | timestamp       | Time in UTC that the query started. Total time includes queuing and execution. with 6 digits of precision for fractional seconds. For example: <b>2009-06-12 11:29:19.131358</b> . |

### Usage notes

You can use the SVCS\_ALERT\_EVENT\_LOG to identify potential issues in your queries, then follow the practices in [Tuning query performance \(p. 463\)](#) to optimize your database design and rewrite your queries. SVCS\_ALERT\_EVENT\_LOG records the following alerts:

- **Missing statistics**

Statistics are missing. Run ANALYZE following data loads or significant updates and use STATUPDATE with COPY operations. For more information, see [Amazon Redshift best practices for designing queries \(p. 21\)](#).

- **Nested loop**

A nested loop is usually a Cartesian product. Evaluate your query to ensure that all participating tables are joined efficiently.

- **Very selective filter**

The ratio of rows returned to rows scanned is less than 0.05. Rows scanned is the value of `rows_pre_user_filter` and rows returned is the value of rows in the [STL\\_SCAN \(p. 1554\)](#) system table. Indicates that the query is scanning an unusually large number of rows to determine the result set. This can be caused by missing or incorrect sort keys. For more information, see [Working with sort keys \(p. 60\)](#).

- **Excessive ghost rows**

A scan skipped a relatively large number of rows that are marked as deleted but not vacuumed, or rows that have been inserted but not committed. For more information, see [Vacuuming tables \(p. 110\)](#).

- **Large distribution**

More than 1,000,000 rows were redistributed for hash join or aggregation. For more information, see [Working with data distribution styles \(p. 49\)](#).

- **Large broadcast**

More than 1,000,000 rows were broadcast for hash join. For more information, see [Working with data distribution styles \(p. 49\)](#).

- **Serial execution**

A DS\_DIST\_ALL\_INNER redistribution style was indicated in the query plan, which forces serial execution because the entire inner table was redistributed to a single node. For more information, see [Working with data distribution styles \(p. 49\)](#).

## Sample queries

The following query shows alert events for four queries.

```
SELECT query, substring(event,0,25) as event,
substring(solution,0,25) as solution,
trim(event_time) as event_time from svcs_alert_event_log order by query;
```

| query    | event                         | solution                      | event_time          |
|----------|-------------------------------|-------------------------------|---------------------|
| 6567     | Missing query planner statist | Run the ANALYZE command       | 2014-01-03 18:20:58 |
| 7450     | Scanned a large number of del | Run the VACUUM command to rec | 2014-01-03 21:19:31 |
| 8406     | Nested Loop Join in the query | Review the join predicates to | 2014-01-04 00:34:22 |
| 29512    | Very selective query filter:r | Review the choice of sort key | 2014-01-06 22:00:00 |
| (4 rows) |                               |                               |                     |

## SVCS\_COMPILE

Records compile time and location for each query segment of queries, including queries run on a scaling cluster as well as queries run on the main cluster.

### Note

System views with the prefix SVCS provide details about queries on both the main and concurrency scaling clusters. The views are similar to the views with the prefix SVL except that the SVL views provide information only for queries run on the main cluster.

SVCS\_COMPILE is visible to all users.

For information about SCL\_COMPILE, see [SVL\\_COMPILE \(p. 1630\)](#).

### Table columns

| Column name | Data type | Description                                                                                            |
|-------------|-----------|--------------------------------------------------------------------------------------------------------|
| userid      | integer   | The ID of the user who generated the entry.                                                            |
| xid         | bigint    | The transaction ID associated with the statement.                                                      |
| pid         | integer   | The process ID associated with the statement.                                                          |
| query       | integer   | The query ID. You can use this ID to join various other system tables and views.                       |
| segment     | integer   | The query segment to be compiled.                                                                      |
| locus       | integer   | The location where the segment runs, <b>1</b> if on a compute node and <b>2</b> if on the leader node. |
| starttime   | timestamp | The time in Universal Coordinated Time (UTC) that the compile started.                                 |
| endtime     | timestamp | The time in UTC that the compile ended.                                                                |
| compile     | integer   | A value that is <b>0</b> if the compile was reused and <b>1</b> if the segment was compiled.           |

### Sample queries

In this example, queries 35878 and 35879 ran the same SQL statement. The compile column for query 35878 shows 1 for four query segments, which indicates that the segments were compiled. Query 35879 shows 0 in the compile column for every segment, indicating that the segments did not need to be compiled again.

```
select userid, xid, pid, query, segment, locus,
datediff(ms, starttime, endtime) as duration, compile
from svcs_compile
where query = 35878 or query = 35879
order by query, segment;
```

| userid | xid    | pid   | query | segment | locus | duration | compile |
|--------|--------|-------|-------|---------|-------|----------|---------|
| 100    | 112780 | 23028 | 35878 | 0       | 1     | 0        | 0       |
| 100    | 112780 | 23028 | 35878 | 1       | 1     | 0        | 0       |
| 100    | 112780 | 23028 | 35878 | 2       | 1     | 0        | 0       |
| 100    | 112780 | 23028 | 35878 | 3       | 1     | 0        | 0       |
| 100    | 112780 | 23028 | 35878 | 4       | 1     | 0        | 0       |
| 100    | 112780 | 23028 | 35878 | 5       | 1     | 0        | 0       |
| 100    | 112780 | 23028 | 35878 | 6       | 1     | 1380     | 1       |
| 100    | 112780 | 23028 | 35878 | 7       | 1     | 1085     | 1       |
| 100    | 112780 | 23028 | 35878 | 8       | 1     | 1197     | 1       |
| 100    | 112780 | 23028 | 35878 | 9       | 2     | 905      | 1       |

|     |        |       |       |   |   |   |   |
|-----|--------|-------|-------|---|---|---|---|
| 100 | 112782 | 23028 | 35879 | 0 | 1 | 0 | 0 |
| 100 | 112782 | 23028 | 35879 | 1 | 1 | 0 | 0 |
| 100 | 112782 | 23028 | 35879 | 2 | 1 | 0 | 0 |
| 100 | 112782 | 23028 | 35879 | 3 | 1 | 0 | 0 |
| 100 | 112782 | 23028 | 35879 | 4 | 1 | 0 | 0 |
| 100 | 112782 | 23028 | 35879 | 5 | 1 | 0 | 0 |
| 100 | 112782 | 23028 | 35879 | 6 | 1 | 0 | 0 |
| 100 | 112782 | 23028 | 35879 | 7 | 1 | 0 | 0 |
| 100 | 112782 | 23028 | 35879 | 8 | 1 | 0 | 0 |
| 100 | 112782 | 23028 | 35879 | 9 | 2 | 0 | 0 |

(20 rows)

## SVCS\_CONCURRENCY\_SCALING\_USAGE

Records the usage periods for concurrency scaling. Each usage period is a consecutive duration where an individual concurrency scaling cluster is actively processing queries.

By default, this view is visible by a superuser only. The database superuser can choose to open it up to all users.

### Table columns

| Column name      | Data type                   | Description                                     |
|------------------|-----------------------------|-------------------------------------------------|
| start_time       | timestamp without time zone | When the usage period starts.                   |
| end_time         | timestamp without time zone | When the usage period ends.                     |
| queries          | bigint                      | Number of queries run during this usage period. |
| usage_in_seconds | decimal(27,0)               | Total seconds in this usage period.             |

### Sample queries

To view the usage duration in seconds for a specific period, enter the following query:

```
select * from svcs_concurrency_scaling_usage order by start_time;
+-----+-----+-----+-----+
| start_time | end_time | queries | usage_in_seconds |
+-----+-----+-----+-----+
| 2019-02-14 18:43:53.01063 | 2019-02-14 19:16:49.781649 | 48 | 1977
```

## SVCS\_EXPLAIN

Displays the EXPLAIN plan for a query that has been submitted for execution.

### Note

System views with the prefix SVCS provide details about queries on both the main and concurrency scaling clusters. The views are similar to the tables with the prefix STL except that the STL tables provide information only for queries run on the main cluster.

This table is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type      | Description                                                                                                                                                      |
|-------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer        | ID of the user who generated the entry.                                                                                                                          |
| query       | integer        | Query ID. The query column can be used to join other system tables and views.                                                                                    |
| nodeid      | integer        | Plan node identifier, where a node maps to one or more steps in the execution of the query.                                                                      |
| parentid    | integer        | Plan node identifier for a parent node. A parent node has some number of child nodes. For example, a merge join is the parent of the scans on the joined tables. |
| plannode    | character(400) | The node text from the EXPLAIN output. Plan nodes that refer to execution on compute nodes are prefixed with <b>XN</b> in the EXPLAIN output.                    |
| info        | character(400) | Qualifier and filter information for the plan node. For example, join conditions and WHERE clause restrictions are included in this column.                      |

## Sample queries

Consider the following EXPLAIN output for an aggregate join query:

```
explain select avg(datediff(day, listtime, saletime)) as avgwait
from sales, listing where sales.listid = listing.listid;
 QUERY PLAN

XN Aggregate (cost=6350.30..6350.31 rows=1 width=16)
 -> XN Hash Join DS_DIST_NONE (cost=47.08..6340.89 rows=3766 width=16)
 Hash Cond: ("outer".listid = "inner".listid)
 -> XN Seq Scan on listing (cost=0.00..1924.97 rows=192497 width=12)
 -> XN Hash (cost=37.66..37.66 rows=3766 width=12)
 -> XN Seq Scan on sales (cost=0.00..37.66 rows=3766 width=12)
(6 rows)
```

If you run this query and its query ID is 10, you can use the SVCS\_EXPLAIN table to see the same kind of information that the EXPLAIN command returns:

```
select query,nodeid,parentid,substring(plannode from 1 for 30),
substring(info from 1 for 20) from svcs_explain
where query=10 order by 1,2;

query| nodeid |parentid| substring | substring
-----+-----+-----+
10 | 1 | 0 | XN Aggregate (cost=6717.61..6 |
10 | 2 | 1 | -> XN Merge Join DS_DIST_NO| Merge Cond:("outer"
10 | 3 | 2 | -> XN Seq Scan on lis |
10 | 4 | 2 | -> XN Seq Scan on sal |
```

(4 rows)

Consider the following query:

```
select event.eventid, sum(pricepaid)
from event, sales
where event.eventid=sales.eventid
group by event.eventid order by 2 desc;

eventid | sum
-----+-----
 289 | 51846.00
 7895 | 51049.00
 1602 | 50301.00
 851 | 49956.00
 7315 | 49823.00
...
```

If this query's ID is 15, the following system table query returns the plan nodes that were performed. In this case, the order of the nodes is reversed to show the actual order of execution:

```
select query,nodeid,parentid,substring(plannode from 1 for 56)
from svcs_explain where query=15 order by 1, 2 desc;

query|nodeid|parentid| substring
-----+-----+-----+
15 | 8 | 7 | -> XN Seq Scan on eve
15 | 7 | 5 | -> XN Hash(cost=87.98..87.9
15 | 6 | 5 | -> XN Seq Scan on sales(cost
15 | 5 | 4 | -> XN Hash Join DS_DIST_OUTER(cost
15 | 4 | 3 | -> XN HashAggregate(cost=862286577.07..
15 | 3 | 2 | -> XN Sort(cost=1000862287175.47..10008622871
15 | 2 | 1 | -> XN Network(cost=1000862287175.47..1000862287197.
15 | 1 | 0 | XN Merge(cost=1000862287175.47..1000862287197.46 rows=87
(8 rows)
```

The following query retrieves the query IDs for any query plans that contain a window function:

```
select query, trim(plannode) from svcs_explain
where plannode like '%Window%';

query| btrim
-----+
26 | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
27 | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
(2 rows)
```

## SVCS\_PLAN\_INFO

Use the SVCS\_PLAN\_INFO table to look at the EXPLAIN output for a query in terms of a set of rows. This is an alternative way to look at query plans.

### Note

System views with the prefix SVCS provide details about queries on both the main and concurrency scaling clusters. The views are similar to the tables with the prefix STL except that the STL tables provide information only for queries run on the main cluster.

This table is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type        | Description                                                                                                                                                                     |
|-------------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer          | ID of the user who generated the entry.                                                                                                                                         |
| query       | integer          | Query ID. The query column can be used to join other system tables and views.                                                                                                   |
| nodeid      | integer          | Plan node identifier, where a node maps to one or more steps in the execution of the query.                                                                                     |
| segment     | integer          | Number that identifies the query segment.                                                                                                                                       |
| step        | integer          | Number that identifies the query step.                                                                                                                                          |
| locus       | integer          | Location where the step runs. 0 if on a compute node and 1 if on the leader node.                                                                                               |
| plannode    | integer          | Enumerated value of the plan node. See the following table for enums for plannode. (The PLANNODE column in <a href="#">SVCS_EXPLAIN (p. 1614)</a> contains the plan node text.) |
| startupcost | double precision | The estimated relative cost of returning the first row for this step.                                                                                                           |
| totalcost   | double precision | The estimated relative cost of executing the step.                                                                                                                              |
| rows        | bigint           | The estimated number of rows that will be produced by the step.                                                                                                                 |
| bytes       | bigint           | The estimated number of bytes that will be produced by the step.                                                                                                                |

## Sample queries

The following examples compare the query plans for a simple SELECT query returned by using the EXPLAIN command and by querying the SVCS\_PLAN\_INFO table.

```

explain select * from category;
QUERY PLAN

XN Seq Scan on category (cost=0.00..0.11 rows=11 width=49)
(1 row)

select * from category;
catid | catgroup | catname | catdesc
-----+-----+-----+
1 | Sports | MLB | Major League Baseball
3 | Sports | NFL | National Football League
5 | Sports | MLS | Major League Soccer
...
select * from svcs_plan_info where query=256;

query | nodeid | segment | step | locus | plannode | startupcost | totalcost
| rows | bytes
-----+-----+-----+-----+-----+-----+-----+-----+
256 | 1 | 0 | 1 | 0 | 104 | 0 | 0.11 | 11 | 539

```

|                                                 |
|-------------------------------------------------|
| 256   1   0   0   0   104   0   0.11   11   539 |
| (2 rows)                                        |

In this example, PLANNODE 104 refers to the sequential scan of the CATEGORY table.

```

select distinct eventname from event order by 1;

eventname

.38 Special
3 Doors Down
70s Soul Jam
A Bronx Tale
...
explain select distinct eventname from event order by 1;

QUERY PLAN

XN Merge (cost=1000000000136.38..1000000000137.82 rows=576 width=17)
Merge Key: eventname
-> XN Network (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Send to leader
-> XN Sort (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Sort Key: eventname
-> XN Unique (cost=0.00..109.98 rows=576 width=17)
-> XN Seq Scan on event (cost=0.00..87.98 rows=8798
width=17)
(8 rows)

select * from svcs_plan_info where query=240 order by nodeid desc;

query | nodeid | segment | step | locus | plannode | startupcost |
totalcost | rows | bytes
-----+-----+-----+-----+-----+-----+-----+
-----+-----+
240 | 5 | 0 | 0 | 0 | 104 | 0 | 87.98 | 8798 | 149566
240 | 5 | 0 | 1 | 0 | 104 | 0 | 87.98 | 8798 | 149566
240 | 4 | 0 | 2 | 0 | 117 | 0 | 109.975 | 576 | 9792
240 | 4 | 0 | 3 | 0 | 117 | 0 | 109.975 | 576 | 9792
240 | 4 | 1 | 0 | 0 | 117 | 0 | 109.975 | 576 | 9792
240 | 4 | 1 | 1 | 0 | 117 | 0 | 109.975 | 576 | 9792
240 | 3 | 1 | 2 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 3 | 2 | 0 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 2 | 2 | 1 | 0 | 123 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 1 | 3 | 0 | 0 | 122 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
(10 rows)

```

## SVCS\_QUERY\_SUMMARY

Use the SVCS\_QUERY\_SUMMARY view to find general information about the execution of a query.

Note that the information in SVCS\_QUERY\_SUMMARY is aggregated from all nodes.

### Note

The SVCS\_QUERY\_SUMMARY view only contains information about queries completed by Amazon Redshift, not other utility and DDL commands. For a complete listing and information on all statements completed by Amazon Redshift, including DDL and utility commands, you can query the SVL\_STATEMENTTEXT view.

System views with the prefix SVCS provide details about queries on both the main and concurrency scaling clusters. The views are similar to the views with the prefix SVL except that the SVL views provide information only for queries run on the main cluster.

SVCS\_QUERY\_SUMMARY is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

For information about SVL\_QUERY\_SUMMARY, see [SVL\\_QUERY\\_SUMMARY \(p. 1647\)](#).

## Table columns

| Column name     | Data type        | Description                                                                                                                                                                                                                                                                                                |
|-----------------|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid          | integer          | ID of user who generated entry.                                                                                                                                                                                                                                                                            |
| query           | integer          | Query ID. Can be used to join various other system tables and views.                                                                                                                                                                                                                                       |
| stm             | integer          | Stream: A set of concurrent segments in a query. A query has one or more streams.                                                                                                                                                                                                                          |
| seg             | integer          | Segment number. A query consists of multiple segments, and each segment consists of one or more steps. Query segments can run in parallel. Each segment runs in a single process.                                                                                                                          |
| step            | integer          | Query step that ran.                                                                                                                                                                                                                                                                                       |
| maxtime         | bigint           | Maximum amount of time for the step to run (in microseconds).                                                                                                                                                                                                                                              |
| avgtime         | bigint           | Average time for the step to run (in microseconds).                                                                                                                                                                                                                                                        |
| rows            | bigint           | Number of data rows involved in the query step.                                                                                                                                                                                                                                                            |
| bytes           | bigint           | Number of data bytes involved in the query step.                                                                                                                                                                                                                                                           |
| rate_row        | double precision | Query execution rate per row.                                                                                                                                                                                                                                                                              |
| rate_byte       | double precision | Query execution rate per byte.                                                                                                                                                                                                                                                                             |
| label           | text             | Step label, which consists of a query step name and, when applicable, table ID and table name (for example, scan <code>tbl=100448 name=user</code> ). Three-digit table IDs usually refer to scans of transient tables. When you see <code>tbl=0</code> , it usually refers to a scan of a constant value. |
| is_diskbased    | character(1)     | Whether this step of the query was performed as a disk-based operation on any node in the cluster: true ( <b>t</b> ) or false ( <b>f</b> ). Only certain steps, such as hash, sort, and aggregate steps, can go to disk. Many types of steps are always run in memory.                                     |
| workmem         | bigint           | Amount of working memory (in bytes) assigned to the query step.                                                                                                                                                                                                                                            |
| is_rrscan       | character(1)     | If true ( <b>t</b> ), indicates that range-restricted scan was used on the step. Default is false ( <b>f</b> ).                                                                                                                                                                                            |
| is_delayed_scan | character(1)     | If true ( <b>t</b> ), indicates that delayed scan was used on the step. Default is false ( <b>f</b> ).                                                                                                                                                                                                     |
| rows_pre_filter | bigint           | For scans of permanent tables, the total number of rows emitted before filtering rows marked for deletion (ghost rows).                                                                                                                                                                                    |

## Sample queries

### Viewing processing information for a query step

The following query shows basic processing information for each step of query 87:

```
select query, stm, seg, step, rows, bytes
from svcs_query_summary
where query = 87
order by query, seg, step;
```

This query retrieves the processing information about query 87, as shown in the following sample output:

| query     | stm | seg | step | rows   | bytes   |
|-----------|-----|-----|------|--------|---------|
| 87        | 0   | 0   | 0    | 90     | 1890    |
| 87        | 0   | 0   | 2    | 90     | 360     |
| 87        | 0   | 1   | 0    | 90     | 360     |
| 87        | 0   | 1   | 2    | 90     | 1440    |
| 87        | 1   | 2   | 0    | 210494 | 4209880 |
| 87        | 1   | 2   | 3    | 89500  | 0       |
| 87        | 1   | 2   | 6    | 4      | 96      |
| 87        | 2   | 3   | 0    | 4      | 96      |
| 87        | 2   | 3   | 1    | 4      | 96      |
| 87        | 2   | 4   | 0    | 4      | 96      |
| 87        | 2   | 4   | 1    | 1      | 24      |
| 87        | 3   | 5   | 0    | 1      | 24      |
| 87        | 3   | 5   | 4    | 0      | 0       |
| (13 rows) |     |     |      |        |         |

### Determining whether query steps spilled to disk

The following query shows whether or not any of the steps for the query with query ID 1025 (see the [SVL\\_QLOG \(p. 1640\)](#) view to learn how to obtain the query ID for a query) spilled to disk or if the query ran entirely in-memory:

```
select query, step, rows, workmem, label, is_diskbased
from svcs_query_summary
where query = 1025
order by workmem desc;
```

This query returns the following sample output:

| query    | step | rows     | workmem   | label           | is_diskbased |
|----------|------|----------|-----------|-----------------|--------------|
| 1025     | 0    | 16000000 | 141557760 | scan tbl=9      | f            |
| 1025     | 2    | 16000000 | 135266304 | hash tbl=142    | t            |
| 1025     | 0    | 16000000 | 128974848 | scan tbl=116536 | f            |
| 1025     | 2    | 16000000 | 122683392 | dist            | f            |
| (4 rows) |      |          |           |                 |              |

By scanning the values for IS\_DISKBASED, you can see which query steps went to disk. For query 1025, the hash step ran on disk. Steps might run on disk include hash, aggr, and sort steps. To view only disk-based query steps, add **and is\_diskbased = 't'** clause to the SQL statement in the above example.

## SVCS\_S3LIST

Use the SVCS\_S3LIST view to get details about Amazon Redshift Spectrum queries at the segment level. One segment can perform one external table scan. This view is derived from the SVL\_S3LIST system view but doesn't show slice-level for queries run on a concurrency scaling cluster.

### Note

System views with the prefix SVCS provide details about queries on both the main and concurrency scaling clusters. The views are similar to the views with the prefix SVL except that the SVL views provide information only for queries run on the main cluster.

SVCS\_S3LIST is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

For information about SVL\_S3LIST, see [SVL\\_S3LIST \(p. 1649\)](#).

### Table columns

| Column name      | Data type        | Description                                                |
|------------------|------------------|------------------------------------------------------------|
| query            | integer          | The query ID.                                              |
| segment          | integer          | The segment number. A query consists of multiple segments. |
| node             | integer          | The node number.                                           |
| eventtime        | timestamp        | The time in UTC that the event is recorded.                |
| bucket           | char(256)        | The Amazon S3 bucket name.                                 |
| prefix           | char(256)        | The prefix of the Amazon S3 bucket location.               |
| recursive        | char(1)          | Whether there is recursive scan for subfolders.            |
| retrieved_files  | integer          | The number of listed files.                                |
| max_file_size    | bigint           | The maximum file size among listed files.                  |
| avg_file_size    | double precision | The average file size among listed files.                  |
| generated_splits | integer          | The number of file splits.                                 |
| avg_split_length | double precision | The average length of file splits in bytes.                |
| duration         | bigint           | The duration of file listing, in microseconds.             |

### Sample query

The following example queries SVCS\_S3LIST for the last query performed.

```
select *
from svcs_s3list
where query = pg_last_query_id()
order by query,segment;
```

## SVCS\_S3LOG

Use the SVCS\_S3LOG view to get troubleshooting details about Redshift Spectrum queries at the segment level. One segment can perform one external table scan. This view is derived from the SVL\_S3LOG system view but doesn't show slice-level for queries run on a concurrency scaling cluster.

**Note**

System views with the prefix SVCS provide details about queries on both the main and concurrency scaling clusters. The views are similar to the views with the prefix SVL except that the SVL views provide information only for queries run on the main cluster.

SVCS\_S3LOG is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

For information about SVL\_S3LOG, see [SVL\\_S3LOG \(p. 1650\)](#).

### Table columns

| Column name | Data type | Description                                                                                                |
|-------------|-----------|------------------------------------------------------------------------------------------------------------|
| pid         | integer   | The process ID.                                                                                            |
| query       | integer   | The query ID.                                                                                              |
| segment     | integer   | The segment number. A query consists of multiple segments, and each segment consists of one or more steps. |
| step        | integer   | The query step that ran.                                                                                   |
| node        | integer   | The node number.                                                                                           |
| eventtime   | timestamp | The time in UTC that the event is recorded.                                                                |
| message     | char(512) | The message for the log entry.                                                                             |

### Sample query

The following example queries SVCS\_S3LOG for the last query that ran.

```
select *
from svcs_s3log
where query = pg_last_query_id()
order by query, segment;
```

## SVCS\_S3PARTITION\_SUMMARY

Use the SVCS\_S3PARTITION\_SUMMARY view to get a summary of Redshift Spectrum queries partition processing at the segment level. One segment can perform one external table scan.

**Note**

System views with the prefix SVCS provide details about queries on both the main and concurrency scaling clusters. The views are similar to the views with the prefix SVL except that the SVL views provide information only for queries run on the main cluster.

SVCS\_S3PARTITION\_SUMMARY is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

For information about SVL\_S3PARTITION, see [SVL\\_S3PARTITION \(p. 1651\)](#).

## Table columns

| Column name             | Data type | Description                                                                            |
|-------------------------|-----------|----------------------------------------------------------------------------------------|
| query                   | integer   | The query ID. You can use this value to join various other system tables and views.    |
| segment                 | integer   | The segment number. A query consists of multiple segments.                             |
| assignment              | char(1)   | The type of partition assignment across nodes.                                         |
| min_starttime           | timestamp | The time in UTC that the partition processing started.                                 |
| max_endtime             | timestamp | The time in UTC that the partition processing completed.                               |
| min_duration            | bigint    | The minimum partition processing time used by a node for this query (in microseconds). |
| max_duration            | bigint    | The maximum partition processing time used by a node for this query (in microseconds). |
| avg_duration            | bigint    | The average partition processing time used by a node for this query (in microseconds). |
| total_partitions        | integer   | The total number of partitions in an external table.                                   |
| qualified_partitions    | integer   | The total number of qualified partitions.                                              |
| min_assigned_partitions | integer   | The minimum number of partitions assigned on one node.                                 |
| max_assigned_partitions | integer   | The maximum number of partitions assigned on one node.                                 |
| avg_assigned_partitions | integer   | The average number of partitions assigned on one node.                                 |

## Sample query

The following example gets the partition scan details for the last query performed.

```
select query, segment, assignment, min_starttime, max_endtime, min_dureation, avg_duration
from svcs_s3partition_summary
where query = pg_last_query_id()
order by query,segment;
```

## SVCS\_S3QUERY\_SUMMARY

Use the SVCS\_S3QUERY\_SUMMARY view to get a summary of all Redshift Spectrum queries (S3 queries) that have been run on the system. One segment can perform one external table scan.

### Note

System views with the prefix SVCS provide details about queries on both the main and concurrency scaling clusters. The views are similar to the views with the prefix SVL except that the SVL views provide information only for queries run on the main cluster.

SVCS\_S3QUERY\_SUMMARY is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

For information about SVL\_S3QUERY, see [SVL\\_S3QUERY \(p. 1653\)](#).

## Table columns

| Column name            | Data type      | Description                                                                                                                                                       |
|------------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid                 | integer        | The ID of the user that generated the given entry.                                                                                                                |
| query                  | integer        | The query ID. You can use this value to join various other system tables and views.                                                                               |
| xid                    | bigint         | The transaction ID.                                                                                                                                               |
| pid                    | integer        | The process ID.                                                                                                                                                   |
| segment                | integer        | The segment number. A query consists of multiple segments, and each segment consists of one or more steps.                                                        |
| step                   | integer        | The query step that ran.                                                                                                                                          |
| starttime              | timestamp      | The time in UTC that the Redshift Spectrum query in this segment started running. One segment can have one external table scan.                                   |
| endtime                | timestamp      | The time in UTC that the Redshift Spectrum query in this segment completed. One segment can have one external table scan.                                         |
| elapsed                | integer        | The length of time that it took the Redshift Spectrum query in this segment to run (in microseconds).                                                             |
| aborted                | integer        | If a query was stopped by the system or canceled by the user, this column contains <b>1</b> . If the query ran to completion, this column contains <b>0</b> .     |
| external_table_name    | character(136) | The internal format of name of the external name of the table for the external table scan.                                                                        |
| file_format            | character(16)  | The file format of the external table data.                                                                                                                       |
| is_partitioned         | char(1)        | If true ( <b>t</b> ), this column value indicates that the external table is partitioned.                                                                         |
| is_rrscan              | char(1)        | If true ( <b>t</b> ), this column value indicates that a range-restricted scan was applied.                                                                       |
| is_nested              | varchar(1)     | If true ( <b>t</b> ), this column value indicates that the nested column data type is accessed.                                                                   |
| s3_scanned_rows        | bigint         | The number of rows scanned from Amazon S3 and sent to the Redshift Spectrum layer.                                                                                |
| s3_scanned_bytes       | bigint         | The number of bytes scanned from Amazon S3 and sent to the Redshift Spectrum layer, based on compressed data.                                                     |
| s3query_returned_rows  | bigint         | The number of rows returned from the Redshift Spectrum layer to the cluster.                                                                                      |
| s3query_returned_bytes | bigint         | The number of bytes returned from the Redshift Spectrum layer to the cluster. A large amount of data returned to Amazon Redshift might affect system performance. |

| Column name             | Data type | Description                                                                                                                                                                                                                                                                     |
|-------------------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| files                   | integer   | The number of files that were processed for this Redshift Spectrum query. A small number of files limits the benefits of parallel processing.                                                                                                                                   |
| files_max               | integer   | The maximum number of file processed on one slice.                                                                                                                                                                                                                              |
| files_avg               | integer   | The average number of file processed on one slice.                                                                                                                                                                                                                              |
| splits                  | bigint    | The number of splits processed for this segment. The number of splits processed on this slice. With large splitable data files, for example, data files larger than about 512 MB, Redshift Spectrum tries to split the files into multiple S3 requests for parallel processing. |
| splits_max              | integer   | The maximum number of splits processed on this slice.                                                                                                                                                                                                                           |
| splits_avg              | bigint    | The average number of splits processed on this slice.                                                                                                                                                                                                                           |
| total_split_size        | bigint    | The total size of all splits processed.                                                                                                                                                                                                                                         |
| max_split_size          | bigint    | The maximum split size processed, in bytes.                                                                                                                                                                                                                                     |
| avg_split_size          | bigint    | The average split size processed, in bytes.                                                                                                                                                                                                                                     |
| total_retries           | bigint    | The total number of retries for the Redshift Spectrum query in this segment.                                                                                                                                                                                                    |
| max_retries             | integer   | The maximum number of retries for one individual processed file.                                                                                                                                                                                                                |
| max_request_duration    | bigint    | The maximum duration of an individual file request (in microseconds). Long running queries might indicate a bottleneck.                                                                                                                                                         |
| avg_request_duration    | bigint    | The average duration of the file requests (in microseconds).                                                                                                                                                                                                                    |
| max_request_parallelism | integer   | The maximum number of parallel requests at one slice for this Redshift Spectrum query.                                                                                                                                                                                          |
| avg_request_parallelism | integer   | The average number of parallel requests at one slice for this Redshift Spectrum query.                                                                                                                                                                                          |
| total_slowdown          | bigint    | The total number of Amazon S3 requests with a slow down error that occurred during the external table scan.                                                                                                                                                                     |
| max_slowdown            | integer   | The maximum number of Amazon S3 requests with a slow down error that occurred during the external table scan on one slice.                                                                                                                                                      |

## Sample query

The following example gets the scan step details for the last query run.

```
select query, segment, elapsed, s3_scanned_rows, s3_scanned_bytes, s3query_returned_rows,
 s3query_returned_bytes, files
 from svcs_s3query_summary
 where query = pg_last_query_id()
```

```
order by query,segment;
```

| query | segment | elapsed | s3_scanned_rows | s3_scanned_bytes | s3query_returned_rows | s3query_returned_bytes | files |
|-------|---------|---------|-----------------|------------------|-----------------------|------------------------|-------|
| 4587  | 2       | 67811   | 0               | 0                | 0                     | 0                      | 0     |
| 4587  | 2       | 591568  | 170260          | 172462           | 11260097              | 8513                   | 1     |
| 4587  | 2       | 216849  | 0               | 0                | 0                     | 0                      | 0     |
| 4587  | 2       | 216671  | 0               | 0                | 0                     | 0                      | 0     |

## SVCS\_STREAM\_SEGS

Lists the relationship between streams and concurrent segments.

**Note**

System views with the prefix SVCS provide details about queries on both the main and concurrency scaling clusters. The views are similar to the tables with the prefix STL except that the STL tables provide information only for queries run on the main cluster.

This table is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                   |
|-------------|-----------|-------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                       |
| query       | integer   | Query ID. The query column can be used to join other system tables and views. |
| stream      | integer   | The set of concurrent segments of a query.                                    |
| segment     | integer   | Number that identifies the query segment.                                     |

### Sample queries

To view the relationship between streams and concurrent segments for the most recent query, type the following query:

```
select *
from svcs_stream_segs
where query = pg_last_query_id();

query | stream | segment
-----+-----+
10 | 1 | 2
10 | 0 | 0
10 | 2 | 4
10 | 1 | 3
10 | 0 | 1
```

(5 rows)

## SVCS\_UNLOAD\_LOG

Use the SVCS\_UNLOAD\_LOG to get details of UNLOAD operations.

SVCS\_UNLOAD\_LOG records one row for each file created by an UNLOAD statement. For example, if an UNLOAD creates 12 files, SVCS\_UNLOAD\_LOG contains 12 corresponding rows. This view is derived from the STL\_UNLOAD\_LOG system table but doesn't show slice-level for queries run on a concurrency scaling cluster.

**Note**

System views with the prefix SVCS provide details about queries on both the main and concurrency scaling clusters. The views are similar to the tables with the prefix STL except that the STL tables provide information only for queries run on the main cluster.

This table is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name   | Data type       | Description                                         |
|---------------|-----------------|-----------------------------------------------------|
| userid        | integer         | The ID of the user who generated the entry.         |
| query         | integer         | The query ID.                                       |
| pid           | integer         | The process ID associated with the query statement. |
| path          | character(1280) | The complete Amazon S3 object path for the file.    |
| start_time    | timestamp       | The start time for the UNLOAD operation.            |
| end_time      | timestamp       | The end time for the UNLOAD operation.              |
| line_count    | bigint          | The number of lines (rows) unloaded to the file.    |
| transfer_size | bigint          | The number of bytes transferred.                    |
| file_format   | character(10)   | The format of unloaded file.                        |

### Sample query

To get a list of the files that were written to Amazon S3 by an UNLOAD command, you can call an Amazon S3 list operation after the UNLOAD completes; however, depending on how quickly you issue the call, the list might be incomplete because an Amazon S3 list operation is eventually consistent. To get a complete, authoritative list immediately, query SVCS\_UNLOAD\_LOG.

The following query returns the path name for files that were created by an UNLOAD for the last query completed:

```
select query, substring(path,0,40) as path
from svcs_unload_log
where query = pg_last_query_id()
order by path;
```

This command returns the following sample output:

```
query | path
-----+
2320 | s3://my-bucket/venue0000_part_00
2320 | s3://my-bucket/venue0001_part_00
2320 | s3://my-bucket/venue0002_part_00
2320 | s3://my-bucket/venue0003_part_00
(4 rows)
```

## SVL views for main cluster

SVL views are system views in Amazon Redshift that contain references to STL tables and logs for more detailed information.

These views provide quicker and easier access to commonly queried data found in those tables.

### Note

The SVL\_QUERY\_SUMMARY view only contains information about queries run by Amazon Redshift, not other utility and DDL commands. For a complete listing and information on all statements run by Amazon Redshift, including DDL and utility commands, you can query the SVL\_STATEMENTTEXT view

### Topics

- [SVL\\_AUTO\\_WORKER\\_ACTION \(p. 1629\)](#)
- [SVL\\_COMPILE \(p. 1630\)](#)
- [SVL\\_DATASHARE\\_CHANGE\\_LOG \(p. 1631\)](#)
- [SVL\\_DATASHARE\\_CROSS\\_REGION\\_USAGE \(p. 1633\)](#)
- [SVL\\_DATASHARE\\_USAGE\\_CONSUMER \(p. 1634\)](#)
- [SVL\\_DATASHARE\\_USAGE\\_PRODUCER \(p. 1634\)](#)
- [SVL\\_FEDERATED\\_QUERY \(p. 1635\)](#)
- [SVL\\_MULTI\\_STATEMENT\\_VIOLATIONS \(p. 1636\)](#)
- [SVL\\_MV\\_REFRESH\\_STATUS \(p. 1638\)](#)
- [SVL\\_QERROR \(p. 1640\)](#)
- [SVL\\_QLOG \(p. 1640\)](#)
- [SVL\\_QUERY\\_METRICS \(p. 1642\)](#)
- [SVL\\_QUERY\\_METRICS\\_SUMMARY \(p. 1643\)](#)
- [SVL\\_QUERY\\_QUEUE\\_INFO \(p. 1644\)](#)
- [SVL\\_QUERY\\_REPORT \(p. 1645\)](#)
- [SVL\\_QUERY\\_SUMMARY \(p. 1647\)](#)
- [SVL\\_S3LIST \(p. 1649\)](#)
- [SVL\\_S3LOG \(p. 1650\)](#)
- [SVL\\_S3PARTITION \(p. 1651\)](#)
- [SVL\\_S3PARTITION\\_SUMMARY \(p. 1652\)](#)
- [SVL\\_S3QUERY \(p. 1653\)](#)
- [SVL\\_S3QUERY\\_SUMMARY \(p. 1655\)](#)
- [SVL\\_S3RETRIES \(p. 1657\)](#)
- [SVL\\_SPATIAL\\_SIMPLIFY \(p. 1658\)](#)

- [SVL\\_SPECTRUM\\_SCAN\\_ERROR \(p. 1659\)](#)
- [SVL\\_STATEMENTTEXT \(p. 1661\)](#)
- [SVL\\_STORED\\_PROC\\_CALL \(p. 1663\)](#)
- [SVL\\_STORED\\_PROC\\_MESSAGES \(p. 1664\)](#)
- [SVL\\_TERMINATE \(p. 1667\)](#)
- [SVL\\_UDF\\_LOG \(p. 1667\)](#)
- [SVL\\_USER\\_INFO \(p. 1669\)](#)
- [SVL\\_VACUUM\\_PERCENTAGE \(p. 1670\)](#)

## SVL\_AUTO\_WORKER\_ACTION

Records automated actions taken by Amazon Redshift on tables defined for automatic optimization.

SVL\_AUTO\_WORKER\_ACTION is visible to all users.

### Table columns

| Column name    | Data type      | Description                                                                                                                                                                                                                     |
|----------------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| table_id       | integer        | The table identifier.                                                                                                                                                                                                           |
| type           | character(32)  | The type of recommendation. Possible values are distkey and sortkey.                                                                                                                                                            |
| status         | character(128) | The completion status of the recommendation. Possible values are Start, Complete, Skipped, Abort, Checkpoint, and Failed.                                                                                                       |
| eventtime      | timestamp      | The timestamp of the status column.                                                                                                                                                                                             |
| sequence       | integer        | The sequence number of a truncated previous_state value. When a single previous_state contains more than 200 characters, additional rows are logged for that value. Sequence is 0 is the first row, 1 is the second, and so on. |
| previous_state | character(200) | The previous distribution style and sort keys of the table before applying the recommendation. The value is truncated into 200-character increments.                                                                            |

Some examples of values of the status column are as follows:

- Skipped:Table not found.
- Skipped:Recommendation is empty.
- Skipped:Apply sortkey recommendation is disabled.
- Skipped:Retry exceeds the maximum limit for a table.
- Skipped:Table column has changed.
- Abort:This table is not AUTO.
- Abort:This table has been recently converted.
- Abort:This table exceeds table size threshold.
- Abort:This table is already the recommended style.
- Checkpoint: progress **21 . 9963%**.

## Sample queries

In the following example, the rows in the result show actions taken by Amazon Redshift.

```
select table_id, type, status, eventtime, sequence, previous_state
from SVL_AUTO_WORKER_ACTION;
```

| table_id        | type    | status                                               |                |
|-----------------|---------|------------------------------------------------------|----------------|
| eventtime       |         | sequence                                             | previous_state |
| 118082          | sortkey | Start                                                | 2020-08-22     |
| 19:42:20.727049 | 0       |                                                      |                |
| 118078          | sortkey | Start                                                | 2020-08-22     |
| 19:43:54.728819 | 0       |                                                      |                |
| 118082          | sortkey | Start                                                | 2020-08-22     |
| 19:42:52.690264 | 0       |                                                      |                |
| 118072          | sortkey | Start                                                | 2020-08-22     |
| 19:44:14.793572 | 0       |                                                      |                |
| 118082          | sortkey | Failed                                               | 2020-08-22     |
| 19:42:20.728917 | 0       |                                                      |                |
| 118078          | sortkey | Complete                                             | 2020-08-22     |
| 19:43:54.792705 | 0       | SORTKEY: None;                                       |                |
| 118086          | sortkey | Complete                                             | 2020-08-22     |
| 19:42:00.72635  | 0       | SORTKEY: None;                                       |                |
| 118082          | sortkey | Complete                                             | 2020-08-22     |
| 19:43:34.728144 | 0       | SORTKEY: None;                                       |                |
| 118072          | sortkey | Skipped:Retry exceeds the maximum limit for a table. | 2020-08-22     |
| 19:44:46.706155 | 0       |                                                      |                |
| 118086          | sortkey | Start                                                | 2020-08-22     |
| 19:42:00.685255 | 0       |                                                      |                |
| 118082          | sortkey | Start                                                | 2020-08-22     |
| 19:43:34.69531  | 0       |                                                      |                |
| 118072          | sortkey | Start                                                | 2020-08-22     |
| 19:44:46.703331 | 0       |                                                      |                |
| 118082          | sortkey | Checkpoint: progress 14.755079%                      | 2020-08-22     |
| 19:42:52.692828 | 0       |                                                      |                |
| 118072          | sortkey | Failed                                               | 2020-08-22     |
| 19:44:14.796071 | 0       |                                                      |                |
| 116723          | sortkey | Abort:This table is not AUTO.                        | 2020-10-28     |
| 05:12:58.479233 | 0       |                                                      |                |
| 110203          | distkey | Abort:This table is not AUTO.                        | 2020-10-28     |
| 05:45:54.67259  | 0       |                                                      |                |

## SVL\_COMPILE

Records compile time and location for each query segment of queries.

SVL\_COMPILE is visible to all users.

For information about SVCS\_COMPILE, see [SVCS\\_COMPILE \(p. 1612\)](#).

### Table columns

| Column name | Data type | Description                                   |
|-------------|-----------|-----------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.       |
| xid         | bigint    | Transaction ID associated with the statement. |

| Column name | Data type | Description                                                                                        |
|-------------|-----------|----------------------------------------------------------------------------------------------------|
| pid         | integer   | Process ID associated with the statement.                                                          |
| query       | integer   | Query ID. Can be used to join various other system tables and views.                               |
| segment     | integer   | The query segment to be compiled.                                                                  |
| locus       | integer   | Location where the segment runs. <b>1</b> if on a compute node and <b>2</b> if on the leader node. |
| starttime   | timestamp | Time in UTC that the compile started.                                                              |
| endtime     | timestamp | Time in UTC that the compile ended.                                                                |
| compile     | integer   | <b>0</b> if the compile was reused, <b>1</b> if the segment was compiled.                          |

## Sample queries

In this example, queries 35878 and 35879 ran the same SQL statement. The compile column for query 35878 shows 1 for four query segments, which indicates that the segments were compiled. Query 35879 shows 0 in the compile column for every segment, indicating that the segments did not need to be compiled again.

```
select userid, xid, pid, query, segment, locus,
datediff(ms, starttime, endtime) as duration, compile
from svl_compile
where query = 35878 or query = 35879
order by query, segment;
```

| userid | xid    | pid   | query | segment | locus | duration | compile |
|--------|--------|-------|-------|---------|-------|----------|---------|
| 100    | 112780 | 23028 | 35878 | 0       | 1     | 0        | 0       |
| 100    | 112780 | 23028 | 35878 | 1       | 1     | 0        | 0       |
| 100    | 112780 | 23028 | 35878 | 2       | 1     | 0        | 0       |
| 100    | 112780 | 23028 | 35878 | 3       | 1     | 0        | 0       |
| 100    | 112780 | 23028 | 35878 | 4       | 1     | 0        | 0       |
| 100    | 112780 | 23028 | 35878 | 5       | 1     | 0        | 0       |
| 100    | 112780 | 23028 | 35878 | 6       | 1     | 1380     | 1       |
| 100    | 112780 | 23028 | 35878 | 7       | 1     | 1085     | 1       |
| 100    | 112780 | 23028 | 35878 | 8       | 1     | 1197     | 1       |
| 100    | 112780 | 23028 | 35878 | 9       | 2     | 905      | 1       |
| 100    | 112782 | 23028 | 35879 | 0       | 1     | 0        | 0       |
| 100    | 112782 | 23028 | 35879 | 1       | 1     | 0        | 0       |
| 100    | 112782 | 23028 | 35879 | 2       | 1     | 0        | 0       |
| 100    | 112782 | 23028 | 35879 | 3       | 1     | 0        | 0       |
| 100    | 112782 | 23028 | 35879 | 4       | 1     | 0        | 0       |
| 100    | 112782 | 23028 | 35879 | 5       | 1     | 0        | 0       |
| 100    | 112782 | 23028 | 35879 | 6       | 1     | 0        | 0       |
| 100    | 112782 | 23028 | 35879 | 7       | 1     | 0        | 0       |
| 100    | 112782 | 23028 | 35879 | 8       | 1     | 0        | 0       |
| 100    | 112782 | 23028 | 35879 | 9       | 2     | 0        | 0       |

(20 rows)

## SVL\_DATASHARE\_CHANGE\_LOG

Records the consolidated view for tracking changes to datashares on both producer and consumer clusters.

Superusers can see all rows; regular users can see only their own data.

## Table columns

| Column name            | Data type    | Description                                                                                                                                                                                                                                                                                                |
|------------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid                 | integer      | The ID of the user taking the action.                                                                                                                                                                                                                                                                      |
| username               | varchar(128) | The name of the user taking the action.                                                                                                                                                                                                                                                                    |
| pid                    | integer      | The ID of the process.                                                                                                                                                                                                                                                                                     |
| xid                    | bigint       | The ID of the transaction.                                                                                                                                                                                                                                                                                 |
| share_id               | integer      | The ID of the datashare affected.                                                                                                                                                                                                                                                                          |
| share_name             | varchar(128) | The name of the datashare.                                                                                                                                                                                                                                                                                 |
| source_database_id     | integer      | The ID of the database to which the datashare belongs.                                                                                                                                                                                                                                                     |
| source_database_name   | varchar(128) | The name of the database to which the datashare belongs.                                                                                                                                                                                                                                                   |
| consumer_database_id   | integer      | The ID of the database imported from the datashare.                                                                                                                                                                                                                                                        |
| consumer_database_name | varchar(128) | The name of the database imported from the datashare.                                                                                                                                                                                                                                                      |
| arn                    | varchar(192) | The ARN of the resource backing the imported database.                                                                                                                                                                                                                                                     |
| recordtime             | timestamp    | The timestamp of the action.                                                                                                                                                                                                                                                                               |
| action                 | varchar(128) | The action being run. Possible values are CREATE DATASHARE, DROP DATASHARE, GRANT ALTER, REVOKE ALTER, GRANT SHARE, REVOKE SHARE, ALTER ADD, ALTER REMOVE, ALTER SET, GRANT USAGE, REVOKE USAGE, CREATE DATABASE, GRANT or REVOKE USAGE on a shared database, DROP SHARED DATABASE, ALTER SHARED DATABASE. |
| status                 | integer      | The status of the action. Possible values are SUCCESS and ERROR-ERROR CODE.                                                                                                                                                                                                                                |
| share_object_type      | varchar(64)  | The type of database object that was added to or removed from the datashare. Possible values are schemas, tables, columns, functions, and views. This is a field for the producer cluster.                                                                                                                 |
| share_object_id        | integer      | The ID of database object that was added to or removed from the datashare. This is a field for the producer cluster.                                                                                                                                                                                       |
| share_object_name      | varchar(128) | The name of database object that was added to or removed from the datashare. This is a field for the producer cluster.                                                                                                                                                                                     |
| target_user_type       | varchar(16)  | The type of users or groups that a privilege was granted to. This is a field for both the producer and consumer cluster.                                                                                                                                                                                   |
| target_userid          | integer      | The ID of users or groups that a privilege was granted to. This is a field for both the producer and consumer cluster.                                                                                                                                                                                     |
| target_username        | varchar(128) | The name of users or groups that a privilege was granted to. This is a field for both the producer and consumer cluster.                                                                                                                                                                                   |
| consumer_account       | varchar(16)  | The account ID of the data consumer. This is a field for the producer cluster.                                                                                                                                                                                                                             |
| consumer_namespace     | varchar(64)  | The namespace of the data consumer account. This is a field for the producer cluster.                                                                                                                                                                                                                      |

| Column name        | Data type    | Description                                                                                                     |
|--------------------|--------------|-----------------------------------------------------------------------------------------------------------------|
| producer_account   | varchar(16)  | The account ID of the producer account that the datashare belongs to. This is a field for the consumer cluster. |
| producer_namespace | varchar(64)  | The namespace of the product account that the datashare belongs to. This is a field for the consumer cluster.   |
| attribute_name     | varchar(64)  | The name of an attribute of the datashare or shared database.                                                   |
| attribute_value    | varchar(128) | The value of an attribute of the datashare or shared database.                                                  |
| message            | varchar(512) | The error message when an action fails.                                                                         |

## Sample queries

The following example shows a SVL\_DATASHARE\_CHANGE\_LOG view.

```
SELECT DISTINCT action
FROM svl_datashare_change_log
WHERE share_object_name LIKE 'ticket%';

action

"ALTER DATASHARE ADD"
```

## SVL\_DATASHARE\_CROSS\_REGION\_USAGE

Use the SVL\_DATASHARE\_CROSS\_REGION\_USAGE view to get a summary of cross-Region data transferred usage caused by cross-Region datasharing query. SVL\_DATASHARE\_CROSS\_REGION\_USAGE aggregates details at the segment level.

SVL\_DATASHARE\_CROSS\_REGION\_USAGE is visible only to superusers.

### Table columns

| Column name      | Data type | Description                                                                                                       |
|------------------|-----------|-------------------------------------------------------------------------------------------------------------------|
| query            | integer   | The ID of the query. Use this value to join other system tables and views.                                        |
| segment          | bigint    | The number of the segment. A query consists of multiple segments, and each segment consists of one or more steps. |
| start_time       | time      | The time in UTC that the data starts transferring.                                                                |
| end_time         | time      | The time in UTC that the data ends transferring.                                                                  |
| transferred_data | bigint    | The number of bytes of data transferred from a producer Region to a consumer Region.                              |
| source_region    | char(25)  | The producer Region that the query transferred data from.                                                         |
| recordtime       | timestamp | The time when the action is recorded.                                                                             |

## Sample queries

The following example shows a SVL\_DATASHARE\_CROSS\_REGION\_USAGE view.

```
SELECT query, segment, transferred_data, source_region
from svl_datashare_cross_region_usage
where query = pg_last_query_id()
order by query,segment;

query | segment | transferred_data | source_region
-----+-----+-----+
200048 | 2 | 4194304 | us-west-1
200048 | 2 | 4194304 | us-east-2
```

## SVL\_DATASHARE\_USAGE\_CONSUMER

Records the activity and usage of datashares. This view is only relevant on the consumer cluster.

SVL\_DATASHARE\_USAGE\_CONSUMER is visible to all users.

### Table columns

| Column name     | Data type    | Description                                           |
|-----------------|--------------|-------------------------------------------------------|
| userid          | integer      | The ID of the user issuing the request.               |
| pid             | integer      | The ID of the leader process running the query.       |
| xid             | bigint       | The context of the current transaction.               |
| request_id      | varchar(50)  | The unique ID of the requested API call.              |
| request_type    | varchar(25)  | The type of the request made to the producer cluster. |
| transaction_uid | varchar(50)  | The unique ID of the transaction.                     |
| recordtime      | timestamp    | The time when the action is recorded.                 |
| status          | integer      | The status of the requested API call.                 |
| error           | varchar(512) | The message for an error.                             |

### Sample queries

The following example shows a SVL\_DATASHARE\_USAGE\_CONSUMER view.

```
SELECT request_type, status, trim(error) AS error
FROM svl_datashare_usage_consumer

request_type | status | error
-----+-----+-----+
"GET RELATION" | 0 |
```

## SVL\_DATASHARE\_USAGE\_PRODUCER

Records the activity and usage of datashares. This view is only relevant on the producer cluster.

SVL\_DATASHARE\_USAGE\_PRODUCER is visible to all users.

## Table columns

| Column name             | Data type    | Description                                                                                                                 |
|-------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------|
| share_id                | integer      | The object ID (OID) of the datashare.                                                                                       |
| share_name              | varchar(128) | The name of the datashare.                                                                                                  |
| request_id              | varchar(50)  | The unique ID of the requested API call.                                                                                    |
| request_type            | varchar(25)  | The type of the request made to the producer cluster.                                                                       |
| object_type             | varchar(64)  | The type of the object being shared from the datashare. Possible values are schemas, tables, columns, functions, and views. |
| object_oid              | integer      | The ID of the object being shared from the datashare.                                                                       |
| object_name             | varchar(128) | The name of the object being shared from the datashare.                                                                     |
| consumer_account        | varchar(16)  | The account of the consumer account that the datashare is shared to.                                                        |
| consumer_namespace      | varchar(64)  | The namespace of the consumer account that the datashare is shared to.                                                      |
| consumer_transaction_id | varchar(50)  | The unique transaction ID of the statement on the consumer cluster.                                                         |
| recordtime              | timestamp    | The time when the action is recorded.                                                                                       |
| status                  | integer      | The status of the datashare.                                                                                                |
| error                   | varchar(512) | The message for an error.                                                                                                   |

## Sample queries

The following example shows a SVL\_DATASHARE\_USAGE\_PRODUCER view.

```
SELECT DISTINCT request_type
FROM svl_datashare_usage_producer
WHERE object_name LIKE 'tickit%';

request_type

"GET RELATION"
```

## SVL\_FEDERATED\_QUERY

Use the SVL\_FEDERATED\_QUERY view to view information about a federated query call.

SVL\_FEDERATED\_QUERY is visible to all users. Superusers can see all rows. Regular users can see only their own data.

## Table columns

| Column name | Data type | Description                           |
|-------------|-----------|---------------------------------------|
| userid      | integer   | The ID of the user running the query. |

| Column name | Data type        | Description                                                                                                                                        |
|-------------|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| xid         | bigint           | The transaction ID.                                                                                                                                |
| pid         | integer          | The ID of the leader process running the query.                                                                                                    |
| query       | integer          | The query ID of a federated call.                                                                                                                  |
| sourcetype  | character (32)   | The federated call source type, for example "PG".                                                                                                  |
| recordtime  | timestamp        | The time when a query is sent for federation. UTC is used.                                                                                         |
| querytext   | character (4000) | The query string sent to the remote PostgreSQL engine for execution.                                                                               |
| num_rows    | bigint           | The number of rows returned by the federated query.                                                                                                |
| num_bytes   | bigint           | The number of bytes returned by the federated query.                                                                                               |
| duration    | bigint           | The time (microseconds) spent fetching rows from cursor calls. It is the time spent running the federated query, as well as, getting results back. |

## Sample queries

To show information about federated query calls, run the following query.

```
select query, trim(sourcetype) as type, recordtime, trim(querytext) as "PG Subquery" from
svl_federated_query where query = 4292;

query | type | recordtime | pg subquery
-----+-----+-----+
+-----+
 4292 | PG | 2020-03-27 04:29:58.485126 | SELECT "level" FROM functional.employees WHERE
 ("level" >= 6)
(1 row)
```

## SVL\_MULTI\_STATEMENT\_VIOLATIONS

Use the SVL\_MULTI\_STATEMENT\_VIOLATIONS view to get a complete record of all of the SQL commands run on the system that violates transaction block restrictions.

Violations occur when you run any of the following SQL commands that Amazon Redshift restricts inside a transaction block or multi-statement requests:

- [CREATE DATABASE \(p. 745\)](#)
- [DROP DATABASE \(p. 863\)](#)
- [ALTER TABLE APPEND \(p. 656\)](#)
- [CREATE EXTERNAL TABLE \(p. 768\)](#)

- DROP EXTERNAL TABLE
- RENAME EXTERNAL TABLE
- ALTER EXTERNAL TABLE
- CREATE TABLESPACE
- DROP TABLESPACE
- [CREATE LIBRARY \(p. 793\)](#)
- [DROP LIBRARY \(p. 867\)](#)
- REBUILDCAT
- INDEXCAT
- REINDEX DATABASE
- [VACUUM \(p. 1015\)](#)
- [GRANT \(p. 887\)](#)
- [COPY \(p. 677\)](#)

**Note**

If there are any entries in this view, then change your corresponding applications and SQL scripts. We recommend changing your application code to move the use of these restricted SQL commands outside of the transaction block. If you need further assistance, contact AWS Support.

SVL\_MULTI\_STATEMENT\_VIOLATIONS is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type      | Description                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer        | The ID of the user who caused the violation.                                                                                                                                                                                                                                                                                                                                                                               |
| database    | character(32)  | The name of the database that the user was connected to.                                                                                                                                                                                                                                                                                                                                                                   |
| cmdname     | character(20)  | The name of the command that cannot run inside a transaction block or multi-statement request. For example, CREATE DATABASE, DROP DATABASE, ALTER TABLE APPEND, CREATE EXTERNAL TABLE, DROP EXTERNAL TABLE, RENAME EXTERNAL TABLE, ALTER EXTERNAL TABLE, CREATE LIBRARY, DROP LIBRARY, REBUILDCAT, INDEXCAT, REINDEX DATABASE, VACUUM, GRANT on external resources, CLUSTER, COPY, CREATE TABLESPACE, and DROP TABLESPACE. |
| xid         | bigint         | The transaction ID associated with the statement.                                                                                                                                                                                                                                                                                                                                                                          |
| pid         | integer        | The process ID for the statement.                                                                                                                                                                                                                                                                                                                                                                                          |
| label       | character(320) | Either the name of the file used to run the query or a label defined with a SET QUERY_GROUP command. If the query is not file-based or the QUERY_GROUP parameter is not set, this field is blank.                                                                                                                                                                                                                          |
| starttime   | timestamp      | The exact time when the statement started executing, with 6 digits of precision for fractional seconds, for example:<br><b>2009-06-12 11:29:19.131358</b>                                                                                                                                                                                                                                                                  |

| Column name | Data type      | Description                                                                                                                                                        |
|-------------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| endtime     | timestamp      | The exact time when the statement finished executing, with 6 digits of precision for fractional seconds, for example:<br><b>2009-06-12 11:29:19.193640</b>         |
| sequence    | integer        | When a single statement contains more than 200 characters, additional rows are logged for that statement. Sequence 0 is the first row, 1 is the second, and so on. |
| type        | varchar(10)    | The type of SQL statement: <b>QUERY</b> , <b>DDL</b> , or <b>UTILITY</b> .                                                                                         |
| text        | character(200) | The SQL text, in 200-character increments. This field might contain special characters such as backslash (\) and newline (\n).                                     |

## Sample query

The following query returns multiple statements that have violations.

```
select * from svl_multi_statement_violations order by starttime asc;

userid | database | cmdname | xid | pid | label | starttime | endtime | sequence | type | text
=====
1 | dev | CREATE DATABASE | 1034 | 5729 |label1 | ***** | ***** | 0 | DDL | create
table c(b int);
1 | dev | CREATE DATABASE | 1034 | 5729 |label1 | ***** | ***** | 0 | UTILITY |
create database b;
1 | dev | CREATE DATABASE | 1034 | 5729 |label1 | ***** | ***** | 0 | UTILITY |
COMMIT
...
```

## SVL\_MV\_REFRESH\_STATUS

The SVL\_MV\_REFRESH\_STATUS view contains a row for the refresh activity of materialized views.

For more information about materialized views, see [Creating materialized views in Amazon Redshift \(p. 198\)](#).

SVL\_MV\_REFRESH\_STATUS is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type | Description                                       |
|-------------|-----------|---------------------------------------------------|
| db_name     | char(128) | The database that contains the materialized view. |
| userid      | bigint    | The ID of the user who performed the refresh.     |
| schema_name | char(128) | The schema of the materialized view.              |
| mv_name     | char(128) | The materialized view name.                       |
| xid         | bigint    | The transaction ID of the refresh.                |

| Column name  | Data type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| starttime    | timestamp | The start time of the refresh.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| endtime      | timestamp | The end time of the refresh.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| status       | text      | <p>The status of the refresh. Example values include the following:</p> <ul style="list-style-type: none"> <li><i>Refresh successfully updated MV incrementally</i><br/>If it's a materialized view for streaming, the message might have additional qualifiers regarding the number of records. These include the following:           <ul style="list-style-type: none"> <li><i>Stream returned no new data</i> – There were no records retrieved.</li> <li><i>All records received from the stream were skipped</i> – Records were retrieved, but due to errors all were skipped.</li> <li><i>Some stream records were skipped</i> – Records were retrieved, but due to errors some were skipped.</li> </ul> </li> </ul> <p>If there are no qualifiers, then at least one record was retrieved and all records are available in the materialized view. There is one remaining possible qualifier:</p> <ul style="list-style-type: none"> <li><i>The stream may contain more data</i> – The refresh ended before Amazon Redshift determined that there were no further records to consume. The stream can be up to date, but it hasn't been confirmed by Amazon Redshift.</li> <li><i>Refresh successfully recomputed MV from scratch</i></li> <li><i>Refresh partially updated MV incrementally up to an active transaction</i></li> <li><i>MV was already updated</i></li> <li><i>Refresh failed. A base table column was renamed</i></li> <li><i>Refresh failed. A base table column type was changed</i></li> <li><i>Refresh failed. A base table was renamed</i></li> <li><i>Refresh failed due to an internal error</i></li> <li><i>Refresh failed. A base table column was dropped</i></li> <li><i>Refresh failed. Schema of MV was renamed</i></li> <li><i>Refresh failed. MV was not found</i></li> <li><i>Auto refresh aborted due to excessive user workload</i></li> <li><i>Refresh failed. Serializable isolation violation</i></li> </ul> |
| refresh_type | char(32)  | The definition of the refresh type. Example values include <i>Manual</i> and <i>Auto</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

## Sample query

To view the refresh status of materialized views, run the following query.

```
select * from svl_mv_refresh_status;
```

This query returns the following sample output:

| db_name | userid | schema    | name    | xid  | starttime                  | endtime                    | status                                          |
|---------|--------|-----------|---------|------|----------------------------|----------------------------|-------------------------------------------------|
| dev     | 169    | mv_schema | mv_test | 6640 | 2020-02-14 02:26:53.497935 | 2020-02-14 02:26:53.556156 | Refresh successfully recomputed MV from scratch |
| dev     | 166    | mv_schema | mv_test | 6517 | 2020-02-14 02:26:39.287438 | 2020-02-14 02:26:39.349539 | Refresh successfully updated MV incrementally   |
| dev     | 162    | mv_schema | mv_test | 6388 | 2020-02-14 02:26:27.863426 | 2020-02-14 02:26:27.918307 | Refresh successfully recomputed MV from scratch |
| dev     | 161    | mv_schema | mv_test | 6323 | 2020-02-14 02:26:20.020717 | 2020-02-14 02:26:20.080002 | Refresh successfully updated MV incrementally   |
| dev     | 161    | mv_schema | mv_test | 6301 | 2020-02-14 02:26:05.796146 | 2020-02-14 02:26:07.853986 | Refresh successfully recomputed MV from scratch |
| dev     | 153    | mv_schema | mv_test | 6024 | 2020-02-14 02:25:18.762335 | 2020-02-14 02:25:20.043462 | MV was already updated                          |
| dev     | 143    | mv_schema | mv_test | 5557 | 2020-02-14 02:24:23.100601 | 2020-02-14 02:24:23.100633 | MV was already updated                          |
| dev     | 141    | mv_schema | mv_test | 5447 | 2020-02-14 02:23:54.102837 | 2020-02-14 02:24:00.310166 | Refresh successfully updated MV incrementally   |
| dev     | 1      | mv_schema | mv_test | 5329 | 2020-02-14 02:22:26.328481 | 2020-02-14 02:22:28.369217 | Refresh successfully recomputed MV from scratch |
| dev     | 138    | mv_schema | mv_test | 5290 | 2020-02-14 02:21:56.885093 | 2020-02-14 02:21:56.885098 | Refresh failed. MV was not found                |

## SVL\_QERROR

The SVL\_QERROR view is deprecated.

## SVL\_QLOG

The SVL\_QLOG view contains a log of all queries run against the database.

Amazon Redshift creates the SVL\_QLOG view as a readable subset of information from the [STL\\_QUERY \(p. 1539\)](#) table. Use this table to find the query ID for a recently run query or to see how long it took a query to complete.

SVL\_QLOG is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                  |
|-------------|-----------|------------------------------------------------------------------------------|
| userid      | integer   | ID of the user who generated the entry.                                      |
| query       | integer   | Query ID. You can use this ID to join various other system tables and views. |

| Column name                    | Data type      | Description                                                                                                                                                                                                                                                                                       |
|--------------------------------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| xid                            | bigint         | Transaction ID.                                                                                                                                                                                                                                                                                   |
| pid                            | integer        | Process ID associated with the query.                                                                                                                                                                                                                                                             |
| starttime                      | timestamp      | Exact time when the statement started executing, with six digits of precision for fractional seconds—for example:<br><b>2009-06-12 11:29:19.131358</b>                                                                                                                                            |
| endtime                        | timestamp      | Exact time when the statement finished executing, with six digits of precision for fractional seconds—for example:<br><b>2009-06-12 11:29:19.193640</b>                                                                                                                                           |
| elapsed                        | bigint         | Length of time that it took the query to run (in microseconds).                                                                                                                                                                                                                                   |
| aborted                        | integer        | If a query was stopped by the system or canceled by the user, this column contains <b>1</b> . If the query ran to completion, this column contains <b>0</b> . Queries that are canceled for workload management purposes and subsequently restarted also have a value of <b>1</b> in this column. |
| label                          | character(320) | Either the name of the file used to run the query, or a label defined with a SET QUERY_GROUP command. If the query is not file-based or the QUERY_GROUP parameter is not set, this field value is default.                                                                                        |
| substring                      | character(60)  | Truncated query text.                                                                                                                                                                                                                                                                             |
| source_query                   | integer        | If the query used result caching, the query ID of the query that was the source of the cached results. If result caching was not used, this field value is NULL.                                                                                                                                  |
| concurrency_scaling_status_txt |                | A description of whether the query ran on the main cluster or concurrency scaling cluster.                                                                                                                                                                                                        |
| from_sp_call                   | integer        | If the query was called from a stored procedure, the query ID of the procedure call. If the query wasn't run as part of a stored procedure, this field is NULL.                                                                                                                                   |

## Sample queries

The following example returns the query ID, execution time, and truncated query text for the five most recent database queries run by the user with `userid = 100`.

```
select query, pid, elapsed, substring from svl_qlog
where userid = 100
order by starttime desc
limit 5;

query | pid | elapsed | substring
-----+----+-----+-----
187752 | 18921 | 18465685 | select query, elapsed, substring from svl_...
204168 | 5117 | 59603 | insert into testtable values (100);
187561 | 17046 | 1003052 | select * from pg_table_def where tablename...
187549 | 17046 | 1108584 | select * from STV_WLM_SERVICE_CLASS_CONFIG
187468 | 17046 | 5670661 | select * from pg_table_def where schemaname...
(5 rows)
```

The following example returns the SQL script name (LABEL column) and elapsed time for a query that was cancelled (**aborted=1**):

```
select query, elapsed, trim(label) querylabel
from svl_qlog where aborted=1;

query | elapsed | querylabel
-----+-----+-----
 16 | 6935292 | alltickittablesjoin.sql
(1 row)
```

## SVL\_QUERY\_METRICS

The SVL\_QUERY\_METRICS view shows the metrics for completed queries. This view is derived from the [STL\\_QUERY\\_METRICS \(p. 1542\)](#) system table. Use the values in this view as an aid to determine threshold values for defining query monitoring rules. For more information, see [WLM query monitoring rules \(p. 527\)](#).

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name          | Data type   | Description                                                                                                                                                                                                                                                           |
|----------------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid               | integer     | ID of the user that ran the query that generated the entry.                                                                                                                                                                                                           |
| query                | integer     | Query ID. The query column can be used to join other system tables and views.                                                                                                                                                                                         |
| service_class        | integer     | ID for the WLM query queue (service class). Query queues are defined in the WLM configuration. Metrics are reported only for user-defined queues. For a list of service class IDs, see <a href="#">WLM service class IDs (p. 534)</a> .                               |
| dimension            | varchar(24) | Dimension on which the metric is reported. Possible values are query, segment, and step.                                                                                                                                                                              |
| segment              | integer     | Segment number. A query consists of multiple segments, and each segment consists of one or more steps. Query segments can run in parallel. Each segment runs in a single process. If the segment value is 0, metrics segment values are rolled up to the query level. |
| step                 | integer     | ID for the type of step performed. The description for the step type is shown in the step_label column..                                                                                                                                                              |
| step_label           | varchar(30) | Type of step performed.                                                                                                                                                                                                                                               |
| query_cpu_time       | bigint      | CPU time used by the query, in seconds. CPU time is distinct from query run time.                                                                                                                                                                                     |
| query_blocks_read    | bigint      | Number of 1 MB blocks read by the query.                                                                                                                                                                                                                              |
| query_execution_time | bigint      | Elapsed execution time for a query, in seconds. Execution time doesn't include time spent waiting in a queue. See query_queue_time for the time queued.                                                                                                               |

| Column name                  | Data type     | Description                                                                                                                                                                                 |
|------------------------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| query_cpu_usage_pebigrntt    | Percent       | Percent of CPU capacity used by the query.                                                                                                                                                  |
| query_temp_blocks_tbigdisk   | bigint        | The amount of disk space used by a query to write intermediate results, in MB.                                                                                                              |
| segment_execution_tbigent    | bigint        | Elapsed execution time for a single segment, in seconds.                                                                                                                                    |
| cpu_skew                     | numeric(38,2) | The ratio of maximum CPU usage for any slice to average CPU usage for all slices. This metric is defined at the segment level.                                                              |
| io_skew                      | numeric(38,2) | The ratio of maximum blocks read (I/O) for any slice to average blocks read for all slices.                                                                                                 |
| scan_row_count               | bigint        | The number of rows in a scan step. The row count is the total number of rows emitted before filtering rows marked for deletion (ghost rows) and before applying user-defined query filters. |
| join_row_count               | bigint        | The number of rows processed in a join step.                                                                                                                                                |
| nested_loop_join_rowbigintnt | bigint        | The number of rows in a nested loop join.                                                                                                                                                   |
| return_row_count             | bigint        | The number of rows returned by the query.                                                                                                                                                   |
| spectrum_scan_row_bogintt    | bigint        | The number of rows scanned by Amazon Redshift Spectrum in Amazon S3.                                                                                                                        |
| spectrum_scan_size_bogintt   | bigint        | The amount of data, in MB, scanned by Amazon Redshift Spectrum in Amazon S3.                                                                                                                |
| query_queue_time             | bigint        | The amount of time in seconds that the query was queued.                                                                                                                                    |

## SVL\_QUERY\_METRICS\_SUMMARY

The SVL\_QUERY\_METRICS\_SUMMARY view shows the maximum values of metrics for completed queries. This view is derived from the [STL\\_QUERY\\_METRICS \(p. 1542\)](#) system table. Use the values in this view as an aid to determine threshold values for defining query monitoring rules. For more information, see [WLM query monitoring rules \(p. 527\)](#).

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name   | Data type | Description                                                                                                                                                                                                                             |
|---------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid        | integer   | ID of the user that ran the query that generated the entry.                                                                                                                                                                             |
| query         | integer   | Query ID. The query column can be used to join other system tables and views.                                                                                                                                                           |
| service_class | integer   | ID for the WLM query queue (service class). Query queues are defined in the WLM configuration. Metrics are reported only for user-defined queues. For a list of service class IDs, see <a href="#">WLM service class IDs (p. 534)</a> . |

| Column name               | Data type     | Description                                                                                                                                                                                 |
|---------------------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| query_cpu_time            | bigint        | CPU time used by the query, in seconds. CPU time is distinct from query run time.                                                                                                           |
| query_blocks_read         | bigint        | Number of 1 MB blocks read by the query.                                                                                                                                                    |
| query_execution_time      | bigint        | Elapsed execution time for a query, in seconds. Execution time doesn't include time spent waiting in a queue.                                                                               |
| query_cpu_usage_percent   | numeric(38,2) | Percent of CPU capacity used by the query.                                                                                                                                                  |
| query_temp_blocks_written | bigint        | The amount of disk space used by a query to write intermediate results, in MB.                                                                                                              |
| segment_execution_time    | bigint        | Elapsed execution time for a single segment, in seconds.                                                                                                                                    |
| cpu_skew                  | numeric(38,2) | The ratio of maximum CPU usage for any slice to average CPU usage for all slices. This metric is defined at the segment level.                                                              |
| io_skew                   | numeric(38,2) | The ratio of maximum blocks read (I/O) for any slice to average blocks read for all slices.                                                                                                 |
| scan_row_count            | bigint        | The number of rows in a scan step. The row count is the total number of rows emitted before filtering rows marked for deletion (ghost rows) and before applying user-defined query filters. |
| join_row_count            | bigint        | The number of rows processed in a join step.                                                                                                                                                |
| nested_loop_join_rows     | bigint        | The number of rows in a nested loop join.                                                                                                                                                   |
| return_row_count          | bigint        | The number of rows returned by the query.                                                                                                                                                   |
| spectrum_scan_row         | bigint        | The number of rows scanned by Amazon Redshift Spectrum in Amazon S3.                                                                                                                        |
| spectrum_scan_size        | bigint        | The amount of data, in MB, scanned by Amazon Redshift Spectrum in Amazon S3.                                                                                                                |
| query_queue_time          | bigint        | The amount of time in seconds that the query was queued.                                                                                                                                    |

## SVL\_QUERY\_QUEUE\_INFO

Summarizes details for queries that spent time in a workload management (WLM) query queue or a commit queue.

The SVL\_QUERY\_QUEUE\_INFO view filters queries performed by the system and shows only queries performed by a user.

The SVL\_QUERY\_QUEUE\_INFO view summarizes information from the [STL\\_QUERY \(p. 1539\)](#), [STL\\_WLM\\_QUERY \(p. 1575\)](#), and [STL\\_COMMIT\\_STATS \(p. 1503\)](#) system tables.

This view is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name          | Data type     | Description                                                                                                   |
|----------------------|---------------|---------------------------------------------------------------------------------------------------------------|
| database             | text          | The name of the database the user was connected to when the query was issued.                                 |
| query                | integer       | Query ID.                                                                                                     |
| xid                  | bigint        | Transaction ID.                                                                                               |
| userid               | integer       | ID of the user that generated the query.                                                                      |
| querytxt             | text          | First 100 characters of the query text.                                                                       |
| queue_start_time     | timestamp     | Time in UTC when the query entered the WLM queue.                                                             |
| exec_start_time      | timestamp     | Time in UTC when query execution started.                                                                     |
| service_class        | integer       | ID for the service class. Service classes are defined in the WLM configuration file.                          |
| slots                | integer       | Number of WLM query slots.                                                                                    |
| queue_elapsed        | bigint        | Time that the query spent waiting in a WLM queue (in seconds).                                                |
| exec_elapsed         | bigint        | Time spent executing the query (in seconds).                                                                  |
| wlm_total_elapsed    | bigint        | Time that the query spent in a WLM queue (queue_elapsed), plus time spent executing the query (exec_elapsed). |
| commit_queue_elapsed | bigint        | Time that the query spent waiting in the commit queue (in seconds).                                           |
| commit_exec_time     | bigint        | Time that the query spent in the commit operation (in seconds).                                               |
| service_class_name   | character(64) | The name of the service class.                                                                                |

## Sample queries

The following example shows the time that queries spent in WLM queues.

```
select query, service_class, queue_elapsed, exec_elapsed, wlm_total_elapsed
from svl_query_queue_info
where wlm_total_elapsed > 0;

query | service_class | queue_elapsed | exec_elapsed | wlm_total_elapsed
-----+-----+-----+-----+-----+
2742669 | 6 | 2 | 916 | 918
2742668 | 6 | 4 | 197 | 201
(2 rows)
```

## SVL\_QUERY\_REPORT

Amazon Redshift creates the SVL\_QUERY\_REPORT view from a UNION of a number of Amazon Redshift STL system tables to provide information about completed query steps.

This view breaks down the information about completed queries by slice and by step, which can help with troubleshooting node and slice issues in the Amazon Redshift cluster.

SVL\_QUERY\_REPORT is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name     | Data type    | Description                                                                                                                                                                                                                                                                                                         |
|-----------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid          | integer      | ID of user who generated entry.                                                                                                                                                                                                                                                                                     |
| query           | integer      | Query ID. Can be used to join various other system tables and views.                                                                                                                                                                                                                                                |
| slice           | integer      | Data slice where the step ran.                                                                                                                                                                                                                                                                                      |
| segment         | integer      | Segment number.<br><br>A query consists of multiple segments, and each segment consists of one or more steps. Query segments can run in parallel. Each segment runs in a single process.                                                                                                                            |
| step            | integer      | Query step that completed.                                                                                                                                                                                                                                                                                          |
| start_time      | timestamp    | Exact time in UTC when the segment started executing, with 6 digits of precision for fractional seconds. For example: <b>2012-12-12 11:29:19.131358</b>                                                                                                                                                             |
| end_time        | timestamp    | Exact time in UTC when the segment finished executing, with 6 digits of precision for fractional seconds. For example: <b>2012-12-12 11:29:19.131467</b>                                                                                                                                                            |
| elapsed_time    | bigint       | Time (in microseconds) that it took the segment to run.                                                                                                                                                                                                                                                             |
| rows            | bigint       | Number of rows produced by the step (per slice). This number represents the number of rows for the slice that result from the execution of the step, not the number of rows received or processed by the step. In other words, this is the number of rows that survive the step and are passed on to the next step. |
| bytes           | bigint       | Number of bytes produced by the step (per slice).                                                                                                                                                                                                                                                                   |
| label           | char(256)    | Step label, which consists of a query step name and, when applicable, table ID and table name (for example, scan <code>tbl=100448 name =user</code> ). Three-digit table IDs usually refer to scans of transient tables. When you see <code>tbl=0</code> , it usually refers to a scan of a constant value.         |
| is_diskbased    | character(1) | Whether this step of the query was performed as a disk-based operation: true ( <b>t</b> ) or false ( <b>f</b> ). Only certain steps, such as hash, sort, and aggregate steps, can go to disk. Many types of steps are always performed in memory.                                                                   |
| workmem         | bigint       | Amount of working memory (in bytes) assigned to the query step. This value is the <code>query_working_mem</code> threshold allocated for use during execution, not the amount of memory that was actually used                                                                                                      |
| is_rrscan       | character(1) | If true ( <b>t</b> ), indicates that range-restricted scan was used on the step.                                                                                                                                                                                                                                    |
| is_delayed_scan | character(1) | If true ( <b>t</b> ), indicates that delayed scan was used on the step.                                                                                                                                                                                                                                             |

| Column name          | Data type | Description                                                                                                                                                            |
|----------------------|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rows_pre_filtebigint |           | For scans of permanent tables, the total number of rows emitted before filtering rows marked for deletion (ghost rows) and before applying user-defined query filters. |

## Sample queries

The following query demonstrates the data skew of the returned rows for the query with query ID 279. Use this query to determine if database data is evenly distributed over the slices in the data warehouse cluster:

```
select query, segment, step, max(rows), min(rows),
case when sum(rows) > 0
then ((cast(max(rows) -min(rows) as float)*count(rows))/sum(rows))
else 0 end
from svl_query_report
where query = 279
group by query, segment, step
order by segment, step;
```

This query should return data similar to the following sample output:

| query     | segment | step | max      | min      | case                 |
|-----------|---------|------|----------|----------|----------------------|
| 279       | 0       | 0    | 19721687 | 19721687 | 0                    |
| 279       | 0       | 1    | 19721687 | 19721687 | 0                    |
| 279       | 1       | 0    | 986085   | 986084   | 1.01411202804304e-06 |
| 279       | 1       | 1    | 986085   | 986084   | 1.01411202804304e-06 |
| 279       | 1       | 4    | 986085   | 986084   | 1.01411202804304e-06 |
| 279       | 2       | 0    | 1775517  | 788460   | 1.00098637606408     |
| 279       | 2       | 2    | 1775517  | 788460   | 1.00098637606408     |
| 279       | 3       | 0    | 1775517  | 788460   | 1.00098637606408     |
| 279       | 3       | 2    | 1775517  | 788460   | 1.00098637606408     |
| 279       | 3       | 3    | 1775517  | 788460   | 1.00098637606408     |
| 279       | 4       | 0    | 1775517  | 788460   | 1.00098637606408     |
| 279       | 4       | 1    | 1775517  | 788460   | 1.00098637606408     |
| 279       | 4       | 2    | 1        | 1        | 0                    |
| 279       | 5       | 0    | 1        | 1        | 0                    |
| 279       | 5       | 1    | 1        | 1        | 0                    |
| 279       | 6       | 0    | 20       | 20       | 0                    |
| 279       | 6       | 1    | 1        | 1        | 0                    |
| 279       | 7       | 0    | 1        | 1        | 0                    |
| 279       | 7       | 1    | 0        | 0        | 0                    |
| (19 rows) |         |      |          |          |                      |

## SVL\_QUERY\_SUMMARY

Use the SVL\_QUERY\_SUMMARY view to find general information about the execution of a query.

The SVL\_QUERY\_SUMMARY view contains a subset of data from the SVL\_QUERY\_REPORT view. Note that the information in SVL\_QUERY\_SUMMARY is aggregated from all nodes.

### Note

The SVL\_QUERY\_SUMMARY view only contains information about queries performed by Amazon Redshift, not other utility and DDL commands. For a complete listing and information on all statements performed by Amazon Redshift, including DDL and utility commands, you can query the SVL\_STATEMENTTEXT view.

SVL\_QUERY\_SUMMARY is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

For information about SVCS\_QUERY\_SUMMARY, see [SVCS\\_QUERY\\_SUMMARY \(p. 1618\)](#).

## Table columns

| Column name     | Data type        | Description                                                                                                                                                                                                                                                                                                |
|-----------------|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid          | integer          | ID of user who generated entry.                                                                                                                                                                                                                                                                            |
| query           | integer          | Query ID. Can be used to join various other system tables and views.                                                                                                                                                                                                                                       |
| stm             | integer          | Stream: A set of concurrent segments in a query. A query has one or more streams.                                                                                                                                                                                                                          |
| seg             | integer          | Segment number. A query consists of multiple segments, and each segment consists of one or more steps. Query segments can run in parallel. Each segment runs in a single process.                                                                                                                          |
| step            | integer          | Query step that ran.                                                                                                                                                                                                                                                                                       |
| maxtime         | bigint           | Maximum amount of time for the step to run (in microseconds).                                                                                                                                                                                                                                              |
| avgtime         | bigint           | Average time for the step to run (in microseconds).                                                                                                                                                                                                                                                        |
| rows            | bigint           | Number of data rows involved in the query step.                                                                                                                                                                                                                                                            |
| bytes           | bigint           | Number of data bytes involved in the query step.                                                                                                                                                                                                                                                           |
| rate_row        | double precision | Query execution rate per row.                                                                                                                                                                                                                                                                              |
| rate_byte       | double precision | Query execution rate per byte.                                                                                                                                                                                                                                                                             |
| label           | text             | Step label, which consists of a query step name and, when applicable, table ID and table name (for example, scan <code>tbl=100448 name=user</code> ). Three-digit table IDs usually refer to scans of transient tables. When you see <code>tbl=0</code> , it usually refers to a scan of a constant value. |
| is_diskbased    | character(1)     | Whether this step of the query was performed as a disk-based operation on any node in the cluster: true ( <code>t</code> ) or false ( <code>f</code> ). Only certain steps, such as hash, sort, and aggregate steps, can go to disk. Many types of steps are always performed in memory.                   |
| workmem         | bigint           | Amount of working memory (in bytes) assigned to the query step.                                                                                                                                                                                                                                            |
| is_rrscan       | character(1)     | If true ( <code>t</code> ), indicates that range-restricted scan was used on the step. Default is false ( <code>f</code> ).                                                                                                                                                                                |
| is_delayed_scan | character(1)     | If true ( <code>t</code> ), indicates that delayed scan was used on the step. Default is false ( <code>f</code> ).                                                                                                                                                                                         |
| rows_pre_filter | bigint           | For scans of permanent tables, the total number of rows emitted before filtering rows marked for deletion (ghost rows).                                                                                                                                                                                    |

## Sample queries

### Viewing processing information for a query step

The following query shows basic processing information for each step of query 87:

```
select query, stm, seg, step, rows, bytes
from svl_query_summary
where query = 87
order by query, seg, step;
```

This query retrieves the processing information about query 87, as shown in the following sample output:

| query     | stm | seg | step | rows   | bytes   |
|-----------|-----|-----|------|--------|---------|
| 87        | 0   | 0   | 0    | 90     | 1890    |
| 87        | 0   | 0   | 2    | 90     | 360     |
| 87        | 0   | 1   | 0    | 90     | 360     |
| 87        | 0   | 1   | 2    | 90     | 1440    |
| 87        | 1   | 2   | 0    | 210494 | 4209880 |
| 87        | 1   | 2   | 3    | 89500  | 0       |
| 87        | 1   | 2   | 6    | 4      | 96      |
| 87        | 2   | 3   | 0    | 4      | 96      |
| 87        | 2   | 3   | 1    | 4      | 96      |
| 87        | 2   | 4   | 0    | 4      | 96      |
| 87        | 2   | 4   | 1    | 1      | 24      |
| 87        | 3   | 5   | 0    | 1      | 24      |
| 87        | 3   | 5   | 4    | 0      | 0       |
| (13 rows) |     |     |      |        |         |

### Determining whether query steps spilled to disk

The following query shows whether or not any of the steps for the query with query ID 1025 (see the [SVL\\_QLOG \(p. 1640\)](#) view to learn how to obtain the query ID for a query) spilled to disk or if the query ran entirely in-memory:

```
select query, step, rows, workmem, label, is_diskbased
from svl_query_summary
where query = 1025
order by workmem desc;
```

This query returns the following sample output:

| query    | step | rows     | workmem   | label           | is_diskbased |
|----------|------|----------|-----------|-----------------|--------------|
| 1025     | 0    | 16000000 | 141557760 | scan tbl=9      | f            |
| 1025     | 2    | 16000000 | 135266304 | hash tbl=142    | t            |
| 1025     | 0    | 16000000 | 128974848 | scan tbl=116536 | f            |
| 1025     | 2    | 16000000 | 122683392 | dist            | f            |
| (4 rows) |      |          |           |                 |              |

By scanning the values for IS\_DISKBASED, you can see which query steps went to disk. For query 1025, the hash step ran on disk. Steps might run on disk include hash, agrgr, and sort steps. To view only disk-based query steps, add **and is\_diskbased = 't'** clause to the SQL statement in the above example.

## SVL\_S3LIST

Use the SVL\_S3LIST view to get details about Amazon Redshift Spectrum queries at the segment level.

SVL\_S3LIST is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name      | Data type        | Description                                                |
|------------------|------------------|------------------------------------------------------------|
| query            | integer          | The query ID.                                              |
| segment          | integer          | The segment number. A query consists of multiple segments. |
| node             | integer          | The node number.                                           |
| slice            | integer          | The data slice that a particular segment ran against.      |
| eventtime        | timestamp        | The time in UTC that the event is recorded.                |
| bucket           | text             | The Amazon S3 bucket name.                                 |
| prefix           | text             | The prefix of the Amazon S3 bucket location.               |
| recursive        | char(1)          | Whether there is recursive scan for subfolders.            |
| retrieved_files  | integer          | The number of listed files.                                |
| max_file_size    | bigint           | The maximum file size among listed files.                  |
| avg_file_size    | double precision | The average file size among listed files.                  |
| generated_splits | integer          | The number of file splits.                                 |
| avg_split_length | double precision | The average length of file splits in bytes.                |
| duration         | bigint           | The duration of file listing, in microseconds.             |

## Sample query

The following example queries SVL\_S3LIST for the last query to run.

```
select *
from svl_s3list
where query = pg_last_query_id()
order by query,segment;
```

## SVL\_S3LOG

Use the SVL\_S3LOG view to get details about Amazon Redshift Spectrum queries at the segment and node slice level.

SVL\_S3LOG is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type | Description                                                                                                |
|-------------|-----------|------------------------------------------------------------------------------------------------------------|
| pid         | integer   | The process ID.                                                                                            |
| query       | integer   | The query ID.                                                                                              |
| segment     | integer   | The segment number. A query consists of multiple segments, and each segment consists of one or more steps. |
| step        | integer   | The query step that ran.                                                                                   |
| node        | integer   | The node number.                                                                                           |
| slice       | integer   | The data slice that a particular segment ran against.                                                      |
| eventtime   | timestamp | Time in UTC that the step started executing.                                                               |
| message     | text      | Message for the log entry.                                                                                 |

## Sample query

The following example queries SVL\_S3LOG for the last query that ran.

```
select *
from svl_s3log
where query = pg_last_query_id()
order by query,segment,slice;
```

## SVL\_S3PARTITION

Use the SVL\_S3PARTITION view to get details about Amazon Redshift Spectrum partitions at the segment and node slice level.

SVL\_S3PARTITION is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type                   | Description                                                                                              |
|-------------|-----------------------------|----------------------------------------------------------------------------------------------------------|
| query       | integer                     | The query ID.                                                                                            |
| segment     | integer                     | A segment number. A query consists of multiple segments, and each segment consists of one or more steps. |
| node        | integer                     | The node number.                                                                                         |
| slice       | integer                     | The data slice that a particular segment ran against.                                                    |
| starttime   | timestamp without time zone | Time in UTC that the partition pruning started executing.                                                |
| endtime     | timestamp without time zone | Time in UTC that the partition pruning completed.                                                        |

| Column name          | Data type | Description                                 |
|----------------------|-----------|---------------------------------------------|
| duration             | bigint    | Elapsed time (in microseconds).             |
| total_partitions     | integer   | Number of total partitions.                 |
| qualified_partitions | integer   | Number of qualified partitions.             |
| assigned_partitions  | integer   | Number of assigned partitions on the slice. |
| assignment           | character | Type of assignment.                         |

## Sample query

The following example gets the partition details for the last query completed.

```
SELECT query, segment,
 MIN(starttime) AS starttime,
 MAX(endtime) AS endtime,
 datediff(ms,MIN(starttime),MAX(endtime)) AS dur_ms,
 MAX(total_partitions) AS total_partitions,
 MAX(qualified_partitions) AS qualified_partitions,
 MAX(assignment) as assignment_type
 FROM svl_s3partition
 WHERE query=pg_last_query_id()
 GROUP BY query, segment
```

| query            | segment              | starttime                  | endtime                    | dur_ms |
|------------------|----------------------|----------------------------|----------------------------|--------|
| total_partitions | qualified_partitions | assignment_type            |                            |        |
| 99232            | 0                    | 2018-04-17 22:43:50.201515 | 2018-04-17 22:43:54.674595 | 4473   |
| 2526             | 334                  | p                          |                            |        |

## SVL\_S3PARTITION\_SUMMARY

Use the SVL\_S3PARTITION\_SUMMARY view to get a summary of Redshift Spectrum queries partition processing at the segment level.

SVL\_S3PARTITION\_SUMMARY is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

For information about SVCS\_S3PARTITION, see [SVCS\\_S3PARTITION\\_SUMMARY \(p. 1622\)](#).

### Table columns

| Column name | Data type | Description                                                                         |
|-------------|-----------|-------------------------------------------------------------------------------------|
| query       | integer   | The query ID. You can use this value to join various other system tables and views. |
| segment     | integer   | The segment number. A query consists of multiple segments.                          |
| assignment  | char(1)   | The type of partition assignment across nodes.                                      |

| Column name             | Data type | Description                                                                            |
|-------------------------|-----------|----------------------------------------------------------------------------------------|
| min_starttime           | timestamp | The time in UTC that the partition processing started.                                 |
| max_endtime             | timestamp | The time in UTC that the partition processing completed.                               |
| min_duration            | bigint    | The minimum partition processing time used by a node for this query (in microseconds). |
| max_duration            | bigint    | The maximum partition processing time used by a node for this query (in microseconds). |
| avg_duration            | bigint    | The average partition processing time used by a node for this query (in microseconds). |
| total_partitions        | integer   | The total number of partitions in an external table.                                   |
| qualified_partitions    | integer   | The total number of qualified partitions.                                              |
| min_assigned_partitions | integer   | The minimum number of partitions assigned on one node.                                 |
| max_assigned_partitions | integer   | The maximum number of partitions assigned on one node.                                 |
| avg_assigned_partitions | integer   | The average number of partitions assigned on one node.                                 |

## Sample query

The following example gets the partition scan details for the last query completed.

```
select query, segment, assignment, min_starttime, max_endtime, min_duration, avg_duration
from svl_s3partition_summary
where query = pg_last_query_id()
order by query,segment;
```

## SVL\_S3QUERY

Use the SVL\_S3QUERY view to get details about Amazon Redshift Spectrum queries at the segment and node slice level.

SVL\_S3QUERY is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                              |
|-------------|-----------|----------------------------------------------------------------------------------------------------------|
| userid      | integer   | The ID of user who generated a given entry.                                                              |
| query       | integer   | The query ID.                                                                                            |
| segment     | integer   | A segment number. A query consists of multiple segments, and each segment consists of one or more steps. |
| step        | integer   | The query step that ran.                                                                                 |
| node        | integer   | The node number.                                                                                         |

| Column name             | Data type        | Description                                                                                                                                                                                                                     |
|-------------------------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| slice                   | integer          | The data slice that a particular segment ran against.                                                                                                                                                                           |
| starttime               | timestamp        | Time in UTC that the query started executing.                                                                                                                                                                                   |
| endtime                 | timestamp        | Time in UTC that the query execution completed                                                                                                                                                                                  |
| elapsed                 | integer          | Elapsed time (in microseconds).                                                                                                                                                                                                 |
| external_table_name     | char(136)        | Internal format of external table name for the s3 scan step.                                                                                                                                                                    |
| is_partitioned          | char(1)          | If true ( <b>t</b> ), this column value indicates that the external table is partitioned.                                                                                                                                       |
| is_rrscan               | char(1)          | If true ( <b>t</b> ), this column value indicates that a range-restricted scan was applied.                                                                                                                                     |
| s3_scanned_rows         | bigrint          | The number of rows scanned from Amazon S3 and sent to the Redshift Spectrum layer.                                                                                                                                              |
| s3_scanned_bytes        | bigin            | The number of bytes scanned from Amazon S3 and sent to the Redshift Spectrum layer.                                                                                                                                             |
| s3query_returned_rows   | bigrint          | The number of rows returned from the Redshift Spectrum layer to the cluster.                                                                                                                                                    |
| s3query_returned_bytes  | bigin            | The number of bytes returned from the Redshift Spectrum layer to the cluster.                                                                                                                                                   |
| files                   | integer          | The number of files that were processed for this S3 scan step on this slice.                                                                                                                                                    |
| splits                  | int              | The number of splits processed on this slice. With large splittable data files, for example, data files larger than about 512 MB, Redshift Spectrum tries to split the files into multiple S3 requests for parallel processing. |
| total_split_size        | bigrint          | The total size of all splits processed on this slice, in bytes.                                                                                                                                                                 |
| max_split_size          | bigrint          | The maximum split size processed for this slice, in bytes.                                                                                                                                                                      |
| total_retries           | integer          | The total number of retries for the processed files.                                                                                                                                                                            |
| max_retries             | integer          | The maximum number of retries for an individual processed file.                                                                                                                                                                 |
| max_request_duration    | integer          | The maximum duration of an individual Redshift Spectrum request (in microseconds).                                                                                                                                              |
| avg_request_duration    | double precision | The average duration of the Redshift Spectrum requests (in microseconds).                                                                                                                                                       |
| max_request_parallelism | integer          | The maximum number of outstanding Redshift Spectrum on this slice for this S3 scan step.                                                                                                                                        |
| avg_request_parallelism | double precision | The average number of parallel Redshift Spectrum requests on this slice for this S3 scan step.                                                                                                                                  |

## Sample query

The following example gets the scan step details for the last query completed.

```
select query, segment, slice, elapsed, s3_scanned_rows, s3_scanned_bytes,
 s3query_returned_rows, s3query_returned_bytes, files
 from svl_s3query
 where query = pg_last_query_id()
 order by query,segment,slice;
```

| query | segment | slice | elapsed | s3_scanned_rows | s3_scanned_bytes | s3query_returned_rows | s3query_returned_bytes | files |
|-------|---------|-------|---------|-----------------|------------------|-----------------------|------------------------|-------|
| 4587  | 2       | 0     | 67811   | 0               | 0                | 0                     | 0                      | 0     |
| 4587  | 2       | 1     | 591568  | 172462          | 11260097         | 170260                | 1                      |       |
| 4587  | 2       | 2     | 216849  | 0               | 0                | 0                     | 0                      | 0     |
| 4587  | 2       | 3     | 216671  | 0               | 0                | 0                     | 0                      | 0     |
| 0     |         |       | 0       | 0               | 0                |                       |                        |       |

## SVL\_S3QUERY\_SUMMARY

Use the SVL\_S3QUERY\_SUMMARY view to get a summary of all Amazon Redshift Spectrum queries (S3 queries) that have been run on the system. SVL\_S3QUERY\_SUMMARY aggregates detail from SVL\_S3QUERY at the segment level.

SVL\_S3QUERY\_SUMMARY is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

For SVCS\_S3QUERY\_SUMMARY, see [SVCS\\_S3QUERY\\_SUMMARY \(p. 1623\)](#).

### Table columns

| Column name | Data type | Description                                                                                                |
|-------------|-----------|------------------------------------------------------------------------------------------------------------|
| userid      | integer   | The ID of the user that generated the given entry.                                                         |
| query       | integer   | The query ID. You can use this value to join various other system tables and views.                        |
| xid         | bigint    | The transaction ID.                                                                                        |
| pid         | integer   | The process ID.                                                                                            |
| segment     | integer   | The segment number. A query consists of multiple segments, and each segment consists of one or more steps. |
| step        | integer   | The query step that ran.                                                                                   |
| starttime   | timestamp | Time in UTC that the query started executing.                                                              |
| endtime     | timestamp | Time in UTC that the query completed.                                                                      |
| elapsed     | integer   | The length of time that it took the query to run (in microseconds).                                        |

| Column name            | Data type     | Description                                                                                                                                                                                                                                                                      |
|------------------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| aborted                | integer       | If a query was stopped by the system or canceled by the user, this column contains <b>1</b> . If the query ran to completion, this column contains <b>0</b> .                                                                                                                    |
| external_table_name    | char(136)     | The internal format of name of the external name of the table for the external table scan.                                                                                                                                                                                       |
| file_format            | character(16) | The file format of the external table data.                                                                                                                                                                                                                                      |
| is_partitioned         | char(1)       | If true ( <b>t</b> ), this column value indicates that the external table is partitioned.                                                                                                                                                                                        |
| is_rrscan              | char(1)       | If true ( <b>t</b> ), this column value indicates that a range-restricted scan was applied.                                                                                                                                                                                      |
| is_nested              | char(1)       | If true ( <b>t</b> ), this column value indicates that the nested column data type is accessed.                                                                                                                                                                                  |
| s3_scanned_rows        | bigint        | The number of rows scanned from Amazon S3 and sent to the Redshift Spectrum layer.                                                                                                                                                                                               |
| s3_scanned_bytes       | bigint        | The number of bytes scanned from Amazon S3 and sent to the Redshift Spectrum layer, based on compressed data.                                                                                                                                                                    |
| s3query_returned_rows  | bigint        | The number of rows returned from the Redshift Spectrum layer to the cluster.                                                                                                                                                                                                     |
| s3query_returned_bytes | bigint        | The number of bytes returned from the Redshift Spectrum layer to the cluster. A large amount of data returned to Amazon Redshift might affect system performance.                                                                                                                |
| files                  | integer       | The number of files that were processed for this Redshift Spectrum query. A small number of files limits the benefits of parallel processing.                                                                                                                                    |
| files_max              | integer       | The maximum number of files processed on one slice.                                                                                                                                                                                                                              |
| files_avg              | integer       | The average number of files processed on one slice.                                                                                                                                                                                                                              |
| splits                 | int           | The number of splits processed for this segment. The number of splits processed on this slice. With large splittable data files, for example, data files larger than about 512 MB, Redshift Spectrum tries to split the files into multiple S3 requests for parallel processing. |
| splits_max             | int           | The maximum number of splits processed on this slice.                                                                                                                                                                                                                            |
| splits_avg             | int           | The average number of splits processed on this slice.                                                                                                                                                                                                                            |
| total_split_size       | bigint        | The total size of all splits processed.                                                                                                                                                                                                                                          |
| max_split_size         | bigint        | The maximum split size processed, in bytes.                                                                                                                                                                                                                                      |
| avg_split_size         | bigint        | The average split size processed, in bytes.                                                                                                                                                                                                                                      |
| total_retries          | integer       | The total number of retries for one individual processed file.                                                                                                                                                                                                                   |
| max_retries            | integer       | The maximum number of retries for any of processed files.                                                                                                                                                                                                                        |

| Column name             | Data type        | Description                                                                                                                |
|-------------------------|------------------|----------------------------------------------------------------------------------------------------------------------------|
| max_request_duration    | integer          | The maximum duration of an individual file request (in microseconds). Long running queries might indicate a bottleneck.    |
| avg_request_duration    | double precision | The average duration of the file requests (in microseconds).                                                               |
| max_request_parallelism | integer          | The maximum number of parallel requests at one slice for this Redshift Spectrum query.                                     |
| avg_request_parallelism | double precision | The average number of parallel requests at one slice for this Redshift Spectrum query.                                     |
| total_slowdown bigint   | bigint           | The total number of Amazon S3 requests with a slow down error that occurred during the external table scan.                |
| max_slowdown integer    | integer          | The maximum number of Amazon S3 requests with a slow down error that occurred during the external table scan on one slice. |

## Sample query

The following example gets the scan step details for the last query completed.

```
select query, segment, elapsed, s3_scanned_rows, s3_scanned_bytes, s3query_returned_rows,
 s3query_returned_bytes, files
 from svl_s3query_summary
 where query = pg_last_query_id()
 order by query,segment;
```

| query | segment | elapsed | s3_scanned_rows | s3_scanned_bytes | s3query_returned_rows | s3query_returned_bytes | files |
|-------|---------|---------|-----------------|------------------|-----------------------|------------------------|-------|
| 4587  | 2       | 67811   | 0               | 0                | 0                     | 0                      |       |
| 4587  | 2       | 591568  | 172462          | 11260097         | 8513                  |                        |       |
| 4587  | 2       | 216849  | 0               | 0                | 0                     | 0                      |       |
| 4587  | 2       | 216671  | 0               | 0                | 0                     | 0                      |       |

## SVL\_S3RETRIES

Use the SVL\_S3RETRIES view to get information about why an Amazon Redshift Spectrum query based on Amazon S3 has failed.

SVL\_S3RETRIES is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name        | Data type                   | Description                                                                                                                                                                              |  |  |
|--------------------|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
| query              | integer                     | The query ID.                                                                                                                                                                            |  |  |
| segment            | integer                     | Segment number.<br><br>A query consists of multiple segments, and each segment consists of one or more steps. Query segments can run in parallel. Each segment runs in a single process. |  |  |
| node               | integer                     | The node number.                                                                                                                                                                         |  |  |
| slice              | integer                     | The data slice that a particular segment ran against.                                                                                                                                    |  |  |
| eventtime          | timestamp without time zone | Time in UTC that the step started executing.                                                                                                                                             |  |  |
| retries            | integer                     | The number of retries for the query.                                                                                                                                                     |  |  |
| successful_fetches | integer                     | The number of times data was returned.                                                                                                                                                   |  |  |
| file_size          | bigint                      | This size of the file.                                                                                                                                                                   |  |  |
| location           | text                        | The location of the table,                                                                                                                                                               |  |  |
| message            | text                        | The error message.                                                                                                                                                                       |  |  |

## Sample query

The following example retrieves data about failed S3 queries.

```
SELECT s3retries.query, s3retries.segment, s3retries.node, s3retries.slice,
 s3retries.eventtime, s3retries.retries,
 s3retries.successful_fetches, s3retries.file_size,
 btrim((s3retries."location")::text) AS "location",
 btrim((s3retries.message)::text)
AS message FROM s3retries;
```

## SVL\_SPATIAL\_SIMPLIFY

You can query the system view SVL\_SPATIAL\_SIMPLIFY to get information about simplified spatial geometry objects using the COPY command. When you use COPY on a shapefile, you can specify SIMPLIFY tolerance, SIMPLIFY AUTO, and SIMPLIFY AUTO max\_tolerance ingestion options. The result of the simplification is summarized in SVL\_SPATIAL\_SIMPLIFY system view.

When SIMPLIFY AUTO max\_tolerance is set, this view contains a row for each geometry that exceeded the maximum size. When SIMPLIFY tolerance is set, then one row for the entire COPY operation is stored. This row references the COPY query ID and the specified simplification tolerance.

SVL\_SPATIAL\_SIMPLIFY is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name       | Data type | Description                                                                                                                                                                                                                                                                             |
|-------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| query             | integer   | The ID of the query (COPY command) that generated this row.                                                                                                                                                                                                                             |
| line_number       | integer   | When COPY SIMPLIFY AUTO option is specified, this value is the record number of the simplified record in the shapefile.                                                                                                                                                                 |
| maximum_tolerance | double    | The distance tolerance value specified in the COPY command. This is either the maximum tolerance value using the SIMPLIFY AUTO option, or the fixed tolerance value using the SIMPLIFY option.                                                                                          |
| initial_size      | integer   | The size in bytes of the GEOMETRY data value before simplification.                                                                                                                                                                                                                     |
| simplified        | char(1)   | When the COPY SIMPLIFY AUTO option is specified, t if the geometry was successfully simplified, or f otherwise. The geometry might not be simplified successfully if after the simplification with the given maximum tolerance its size is still larger than the maximum geometry size. |
| final_size        | integer   | When the COPY SIMPLIFY AUTO option is specified, this is the size in bytes of the geometry after simplification.                                                                                                                                                                        |
| final_tolerance   | double    |                                                                                                                                                                                                                                                                                         |

## Sample query

The following query returns the list of records that COPY simplified.

```
SELECT * FROM svl_spatial_simplify WHERE query = pg_last_copy_id();
query | line_number | maximum_tolerance | initial_size | simplified | final_size | final_tolerance
-----+-----+-----+-----+-----+-----+
20 | 1184704 | -1 | 1513736 | t | 1008808 |
1.276386653895e-05
20 | 1664115 | -1 | 1233456 | t | 1023584 |
6.11707814796635e-06
```

## SVL\_SPECTRUM\_SCAN\_ERROR

You can query the system view SVL\_SPECTRUM\_SCAN\_ERROR to get information about Redshift Spectrum scan errors.

SVL\_SPECTRUM\_SCAN\_ERROR is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

Displays a sample of logged errors. The default is 10 entries per query.

| Column name    | Data type      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid         | integer        | The ID of the user that generated this row.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| query          | integer        | The ID of the query that generated this row.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| location       | character(128) | The location of the data being queried.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| rowid          | character(128) | <p>The location of the error within the file. The <code>rowid</code> parts are separated with a : (colon) and additional parts might be added in the future.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <code>row_offset:row_group:row_id</code> </div> <p>A <code>row_offset</code> is the offset (in bytes) of the row within the file and is set to -1 for unsupported file formats. A table is divided into <code>row_groups</code>, and each group has rows with distinct <code>row_ids</code>.</p> |
| colname        | character(128) | The name of the column returned by the query.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| original_value | character(128) | Original value queried.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| modified_value | character(128) | Modified value returned based on the data handling configuration option specified in the query.                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| trigger        | character(128) | Data handling option specified in the query.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| action         | character(128) | Action associated with the data handling option specified in the query.                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| action_value   | character(128) | Value of action parameter associated with the data handling option specified in the query.                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| error_code     | integer        | Result code of the data handling option specified in the query.                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

## Sample query

The following query returns the list of rows for which data handling operations were performed.

```
SELECT * FROM svl_spectrum_scan_error;
```

The query returns results similar to the following.

| userid | query   | location                                          | original_value          | modified_value       | trigger     | action   | rowid | colname     |
|--------|---------|---------------------------------------------------|-------------------------|----------------------|-------------|----------|-------|-------------|
| 100    | 1574007 | s3://spectrum-uddh/league/spi_global_rankings.0:0 | Barclays Premier League | Barclays Premier Lea | UNSPECIFIED | TRUNCATE | 156   | league_name |

```

100 1574007 s3://spectrum-uddh/league/spi_global_rankings.0:0 league_nspi
34595 32767 UNSPECIFIED OVERFLOW_VALUE
199
100 1574007 s3://spectrum-uddh/league/spi_global_rankings.0:1 league_nspi
34151 32767 UNSPECIFIED OVERFLOW_VALUE
199
100 1574007 s3://spectrum-uddh/league/spi_global_rankings.0:2 league_name
Barclays Premier League Barclays Premier Lea UNSPECIFIED TRUNCATE
156
100 1574007 s3://spectrum-uddh/league/spi_global_rankings.0:2 league_nspi
33223 32767 UNSPECIFIED OVERFLOW_VALUE
199
100 1574007 s3://spectrum-uddh/league/spi_global_rankings.0:3 league_name
Barclays Premier League Barclays Premier Lea UNSPECIFIED TRUNCATE
156
100 1574007 s3://spectrum-uddh/league/spi_global_rankings.0:3 league_nspi
32808 32767 UNSPECIFIED OVERFLOW_VALUE
199
100 1574007 s3://spectrum-uddh/league/spi_global_rankings.0:4 league_nspi
32790 32767 UNSPECIFIED OVERFLOW_VALUE
199
100 1574007 s3://spectrum-uddh/league/spi_global_rankings.0:5 league_name
Spanish Primera Division Spanish Primera Divi UNSPECIFIED TRUNCATE
156
100 1574007 s3://spectrum-uddh/league/spi_global_rankings.0:6 league_name
Spanish Primera Division Spanish Primera Divi UNSPECIFIED TRUNCATE
156

```

## SVL\_STATEMENTTEXT

Use the SVL\_STATEMENTTEXT view to get a complete record of all of the SQL commands that have been run on the system.

The SVL\_STATEMENTTEXT view contains the union of all of the rows in the [STL\\_DDLTEXT \(p. 1507\)](#), [STL\\_QUERYTEXT \(p. 1544\)](#), and [STL\\_UTILITYTEXT \(p. 1567\)](#) tables. This view also includes a join to the STL\_QUERY table.

SVL\_STATEMENTTEXT is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type      | Description                                                                                                                                                                                       |
|-------------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid      | integer        | ID of user who generated entry.                                                                                                                                                                   |
| xid         | bigint         | Transaction ID associated with the statement.                                                                                                                                                     |
| pid         | integer        | Process ID for the statement.                                                                                                                                                                     |
| label       | character(320) | Either the name of the file used to run the query or a label defined with a SET QUERY_GROUP command. If the query is not file-based or the QUERY_GROUP parameter is not set, this field is blank. |
| starttime   | timestamp      | Exact time when the statement started executing, with 6 digits of precision for fractional seconds. For example:<br><b>2009-06-12 11:29:19.131358</b>                                             |

| Column name | Data type      | Description                                                                                                                                                        |
|-------------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| endtime     | timestamp      | Exact time when the statement finished executing, with 6 digits of precision for fractional seconds. For example:<br><b>2009-06-12 11:29:19.193640</b>             |
| sequence    | integer        | When a single statement contains more than 200 characters, additional rows are logged for that statement. Sequence 0 is the first row, 1 is the second, and so on. |
| type        | varchar(10)    | Type of SQL statement: <b>QUERY, DDL, or UTILITY</b> .                                                                                                             |
| text        | character(200) | SQL text, in 200-character increments. This field might contain special characters such as backslash (\) and newline (\n).                                         |

## Sample query

The following query returns DDL statements that were run on June 16th, 2009:

```
select starttime, type, rtrim(text) from svl_statementtext
where starttime like '2009-06-16%' and type='DDL' order by starttime asc;

starttime	type	rtrim
2009-06-16 10:36:50.625097 | DDL | create table ddltest(c1 int);
2009-06-16 15:02:16.006341 | DDL | drop view alltickitjoin;
2009-06-16 15:02:23.65285 | DDL | drop table sales;
2009-06-16 15:02:24.548928 | DDL | drop table listing;
2009-06-16 15:02:25.536655 | DDL | drop table event;
...
```

## Reconstructing stored SQL

To reconstruct the SQL stored in the text column of SVL\_STATEMENTTEXT, run a SELECT statement to create SQL from 1 or more parts in the text column. Before running the reconstructed SQL, replace any (\n) special characters with a new line. The result of the following SELECT statement is rows of reconstructed SQL in the query\_statement field.

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '') within
group (order by sequence) AS query_statement
from SVL_STATEMENTTEXT where pid=pg_backend_pid();
```

For example, the following query selects 3 columns. The query itself is longer than 200 characters and is stored in parts in SVL\_STATEMENTTEXT.

```
select
1 AS a012345678901234567890123456789012345678901234567890,
2 AS b012345678901234567890123456789012345678901234567890,
3 AS b0123456789012345678901234567890123456789012345678901234
FROM stl_querytext;
```

In this example, the query is stored in 2 parts (rows) in the text column of SVL\_STATEMENTTEXT.

```
select sequence, text from SVL_STATEMENTTEXT where pid = pg_backend_pid() order by
starttime, sequence;
```

```

sequence | text

+-----+
 0 | select\n1 AS a012345678901234567890123456789012345678901234567890,
 \n2 AS b0123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b0123456789012345678901234567890123456789012345678901234
 1 | \nFROM stl_querytext;

```

To reconstruct the SQL stored in `STL_STATEMENTTEXT`, run the following SQL.

```

select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '') within
group (order by sequence) AS text
from SVL_STATEMENTTEXT where pid=pg_backend_pid();

```

To use the resulting reconstructed SQL in your client, replace any (\n) special characters with a new line.

```

text

select\n1 AS a012345678901234567890123456789012345678901234567890,
\n2 AS b0123456789012345678901234567890123456789012345678901234567890,\n3 AS
b012345678901234567890123456789012345678901234\nFROM stl_querytext;

```

## SVL\_STORED\_PROC\_CALL

You can query the system view `SVL_STORED_PROC_CALL` to get information about stored procedure calls, including start time, end time, and whether a call is canceled. Each stored procedure call receives a query ID.

`SVL_STORED_PROC_CALL` is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name    | Data type      | Description                                                                                                                                                                                              |
|----------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid         | integer        | The ID of the user whose privileges were used to run the statement. If this call was nested within a SECURITY DEFINER stored procedure, then this is the userid of the owner of that stored procedure.   |
| session_userid | integer        | The ID of the user that created the session and is the invoker of the top-level stored procedure call.                                                                                                   |
| query          | integer        | The query ID of the procedure call.                                                                                                                                                                      |
| label          | character(320) | Either the name of the file used to run the query or a label defined with a SET QUERY_GROUP command. If the query is not file-based or the QUERY_GROUP parameter isn't set, this field value is default. |
| xid            | bigint         | The transaction ID.                                                                                                                                                                                      |

| Column name  | Data type       | Description                                                                                                                                                                                                                                                                                                                                                                  |
|--------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| pid          | integer         | The process ID. Usually, all of the queries in a session are run in the same process, so this value usually remains constant if you run a series of queries in the same session. Following certain internal events, Amazon Redshift might restart an active session and assign a new pid value. For more information, see <a href="#">STL_RESTARTED_SESSIONS (p. 1547)</a> . |
| database     | character(32)   | The name of the database that the user was connected to when the query was issued.                                                                                                                                                                                                                                                                                           |
| querytxt     | character(4000) | The actual text of the procedure call query.                                                                                                                                                                                                                                                                                                                                 |
| starttime    | timestamp       | The time in UTC that the query started running, with six digits of precision for fractional seconds, for example: <b>2009-06-12 11:29:19.131358</b> .                                                                                                                                                                                                                        |
| endtime      | timestamp       | The time in UTC that the query finished running, with six digits of precision for fractional seconds, for example: <b>2009-06-12 11:29:19.131358</b> .                                                                                                                                                                                                                       |
| aborted      | integer         | If a stored procedure was stopped by the system or canceled by the user, this column contains 1. If the call runs to completion, this column contains 0.                                                                                                                                                                                                                     |
| from_sp_call | integer         | If the procedure call was invoked by another procedure call, this column contains the query ID of the outer call. Otherwise, the field is NULL.                                                                                                                                                                                                                              |

## Sample query

The following query returns the elapsed time in descending order and the completion status for stored procedure calls in the past day.

```
select query, datediff(seconds, starttime, endtime) as elapsed_time, aborted,
trim(querytxt) as call from svl_stored_proc_call where starttime >= current_timestamp -
interval '1 day' order by 2 desc;

query | elapsed_time | aborted | call
-----+-----+-----+
4166 | 7 | 0 | call search_batch_status(35, 'succeeded');
2433 | 3 | 0 | call test_batch (123456)
1810 | 1 | 0 | call prod_benchmark (123456)
1836 | 1 | 0 | call prod_testing (123456)
1808 | 1 | 0 | call prod_portfolio ('N', 123456)
1816 | 1 | 1 | call prod_portfolio ('Y', 123456)
(6 rows)
```

## SVL\_STORED\_PROC\_MESSAGES

You can query the system view SVL\_STORED\_PROC\_MESSAGES to get information about stored procedure messages. Raised messages are logged even if the stored procedure call is canceled. Each stored procedure call receives a query ID. For more information about how to set the minimum level for logged messages, see `stored_proc_log_min_messages`.

`SVL_STORED_PROC_MESSAGES` is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views](#).

## Table columns

| Column name    | Data type       | Description                                                                                                                                                                                              |
|----------------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| userid         | integer         | The ID of the user whose privileges were used to run the statement. If this call was nested within a SECURITY DEFINER stored procedure, then this is the userid of the owner of that stored procedure.   |
| session_userid | integer         | The ID of the user that created the session and is the invoker of the top-level stored procedure call.                                                                                                   |
| pid            | integer         | The process ID.                                                                                                                                                                                          |
| xid            | bigint          | The transaction ID.                                                                                                                                                                                      |
| query          | integer         | The query ID of the procedure call.                                                                                                                                                                      |
| recordtime     | timestamp       | The time in UTC that the message was raised.                                                                                                                                                             |
| loglevel       | integer         | The numeric value of the log level of the raised message. Possible values: 20 – for LOG 30 – for INFO 40 – for NOTICE 50 – for WARNING 60 – for EXCEPTION                                                |
| loglevel_text  | character(10)   | The log level that corresponds to the numeric value in loglevel. Possible values: LOG, INFO, NOTICE, WARNING, and EXCEPTION.                                                                             |
| message        | character(1024) | The text of the raised message.                                                                                                                                                                          |
| linenum        | integer         | The line number of the raised statement.                                                                                                                                                                 |
| querytext      | character(500)  | The actual text of the procedure call query.                                                                                                                                                             |
| label          | character(320)  | Either the name of the file used to run the query or a label defined with a SET QUERY_GROUP command. If the query is not file-based or the QUERY_GROUP parameter isn't set, this field value is default. |
| aborted        | integer         | If a stored procedure was stopped by the system or canceled by the user, this column contains 1. If the call runs to completion, this column contains 0.                                                 |

## Sample query

The following SQL statements show how to use `SVL_STORED_PROC_MESSAGES` to review raised messages.

```
-- Create and run a stored procedure
CREATE OR REPLACE PROCEDURE test_proc1(f1 int) AS
$$
BEGIN
 RAISE INFO 'Log Level: Input f1 is %',f1;
 RAISE NOTICE 'Notice Level: Input f1 is %',f1;
```

```

EXECUTE 'select invalid';
RAISE NOTICE 'Should not print this';

EXCEPTION WHEN OTHERS THEN
 raise exception 'EXCEPTION level: Exception Handling';
END;
$$ LANGUAGE plpgsql;

-- Call this stored procedure
CALL test_proc1(2);

-- Show raised messages with level higher than INFO
SELECT query, recordtime, loglevel, loglevel_text, trim(message) as message, aborted FROM
svl_stored_proc_messages
WHERE loglevel > 30 AND query = 193 ORDER BY recordtime;

query | recordtime | loglevel | loglevel_text | message
 | aborted
-----+-----+-----+-----+
+-----+
 193 | 2020-03-17 23:57:18.277196 | 40 | NOTICE | Notice Level: Input f1 is
2 | 1
 193 | 2020-03-17 23:57:18.277987 | 60 | EXCEPTION | EXCEPTION level: Exception
Handling | 1
(2 rows)

-- Show raised messages at EXCEPTION level
SELECT query, recordtime, loglevel, loglevel_text, trim(message) as message, aborted FROM
svl_stored_proc_messages
WHERE loglevel_text = 'EXCEPTION' AND query = 193 ORDER BY recordtime;

query | recordtime | loglevel | loglevel_text | message
 | aborted
-----+-----+-----+-----+
+-----+
 193 | 2020-03-17 23:57:18.277987 | 60 | EXCEPTION | EXCEPTION level: Exception
Handling | 1

```

The following SQL statements show how to use SVL\_STORED\_PROC\_MESSAGES to review raised messages with the SET option when creating a stored procedure. Because test\_proc() has a minimum log level of NOTICE, only NOTICE, WARNING, and EXCEPTION level messages are logged in SVL\_STORED\_PROC\_MESSAGES.

```

-- Create a stored procedure with minimum log level of NOTICE
CREATE OR REPLACE PROCEDURE test_proc() AS
$$
BEGIN
 RAISE LOG 'Raise LOG messages';
 RAISE INFO 'Raise INFO messages';
 RAISE NOTICE 'Raise NOTICE messages';
 RAISE WARNING 'Raise WARNING messages';
 RAISE EXCEPTION 'Raise EXCEPTION messages';
 RAISE WARNING 'Raise WARNING messages again'; -- not reachable
END;
$$ LANGUAGE plpgsql SET stored_proc_log_min_messages = NOTICE;

-- Call this stored procedure
CALL test_proc();

-- Show the raised messages
SELECT query, recordtime, loglevel_text, trim(message) as message, aborted FROM
svl_stored_proc_messages
WHERE query = 149 ORDER BY recordtime;

```

| query    | recordtime                 | loglevel_text | message                  | aborted |
|----------|----------------------------|---------------|--------------------------|---------|
| 149      | 2020-03-16 21:51:54.847627 | NOTICE        | Raise NOTICE messages    | 1       |
| 149      | 2020-03-16 21:51:54.84766  | WARNING       | Raise WARNING messages   | 1       |
| 149      | 2020-03-16 21:51:54.847668 | EXCEPTION     | Raise EXCEPTION messages | 1       |
| (3 rows) |                            |               |                          |         |

## SVL\_TERMINATE

Records the time when a user cancels or terminates a process.

`SELECT PG_TERMINATE_BACKEND(pid)`, `SELECT PG_CANCEL_BACKEND(pid)`, and `CANCEL pid` creates a log entry in `SVL_TERMINATE`.

`SVL_TERMINATE` is visible only to the superuser.

### Table columns

| Column name | Data type | Description                                             |
|-------------|-----------|---------------------------------------------------------|
| pid         | integer   | The process ID of the canceled or terminated process.   |
| eventtime   | timestamp | The time when the process is canceled or terminated.    |
| userid      | integer   | The user ID of the user running the command.            |
| type        | string    | The type of termination. It can be CANCEL or TERMINATE. |

The following command shows the latest cancelled query.

```
select * from svl_terminate order by eventtime desc limit 1;
pid | eventtime | userid | type
-----+-----+-----+-----+
8324 | 2020-03-24 09:42:07.298937 | 1 | CANCEL
(1 row)
```

## SVL\_UDF\_LOG

Records system-defined error and warning messages generating during user-defined function (UDF) execution.

This view is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type  | Description                                                                      |
|-------------|------------|----------------------------------------------------------------------------------|
| query       | bigint     | The query ID. You can use this ID to join various other system tables and views. |
| message     | char(4096) | The message generated by the function.                                           |

| Column name | Data type      | Description                                                                                                                                          |
|-------------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| created     | timestamp      | The time that the log was created.                                                                                                                   |
| traceback   | char(4096)     | If available, this value provides a stack traceback for the UDF. For more information, see <a href="#">traceback</a> in the Python Standard Library. |
| funcname    | character(256) | The name of the UDF that is executing.                                                                                                               |
| node        | integer        | The node where the message was generated.                                                                                                            |
| slice       | integer        | The slice where the message was generated.                                                                                                           |
| seq         | integer        | The sequence of the message on the slice.                                                                                                            |

## Sample queries

The following example shows how UDFs handle system-defined errors. The first block shows the definition for a UDF function that returns the inverse of an argument. When you run the function and provide a 0 argument, as the second block shows, the function returns an error. The third statement reads the error message that is logged in SVL\_UDF\_LOG

```
-- Create a function to find the inverse of a number

CREATE OR REPLACE FUNCTION f_udf_inv(a int)
RETURNS float IMMUTABLE
AS $$
 return 1/a
$$ LANGUAGE plpythonu;

-- Run the function with a 0 argument to create an error
Select f_udf_inv(0) from sales;

-- Query SVL_UDF_LOG to view the message

Select query, created, message::varchar
from svl_udf_log;

query | created | message
-----+-----+
+-----+
 2211 | 2015-08-22 00:11:12.04819 | ZeroDivisionError: long division or modulo by zero
\nNone
```

The following example adds logging and a warning message to the UDF so that a divide by zero operation results in a warning message instead of stopping with an error message.

```
-- Create a function to find the inverse of a number and log a warning

CREATE OR REPLACE FUNCTION f_udf_inv_log(a int)
RETURNS float IMMUTABLE
AS $$
import logging
```

```

logger = logging.getLogger() #get root logger
if a==0:
 logger.warning('You attempted to divide by zero.\nReturning zero instead of error.\n')
 return 0
else:
 return 1/a
$$ LANGUAGE plpythonu;

```

The following example runs the function, then queries SVL\_UDF\_LOG to view the message.

```

-- Run the function with a 0 argument to trigger the warning
Select f_udf_inv_log(0) from sales;

-- Query SVL_UDF_LOG to view the message

Select query, created, message::varchar
from svl_udf_log;

query | created | message
-----+-----+-----+
 0 | 2015-08-22 00:11:12.04819 | You attempted to divide by zero.
 Returning zero instead of error.

```

## SVL\_USER\_INFO

You can retrieve data about Amazon Redshift database users with the SVL\_USER\_INFO view.

### Table columns

| Column name    | Data type | Description                                                                                                                                                                                                                                                                                                                                                  |
|----------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| username       | text      | The user name for the role.                                                                                                                                                                                                                                                                                                                                  |
| usesysid       | integer   | The user ID for the user.                                                                                                                                                                                                                                                                                                                                    |
| usecreatedb    | boolean   | A value that indicates whether the user has permissions to create databases.                                                                                                                                                                                                                                                                                 |
| usesuper       | boolean   | A value that indicates whether the user is a superuser.                                                                                                                                                                                                                                                                                                      |
| usecatupd      | boolean   | A value that indicates whether the user can update system catalogs.                                                                                                                                                                                                                                                                                          |
| useconnlimit   | text      | The number of connections that the user can open.                                                                                                                                                                                                                                                                                                            |
| syslogaccess   | text      | A value that indicates whether the user has access to the system logs. The two possible values are RESTRICTED and UNRESTRICTED. RESTRICTED means that users that are not superusers can see their own records. UNRESTRICTED means that user that are not superusers can see all records in the system views and tables to which they have SELECT privileges. |
| last_ddl_ts    | timestamp | The timestamp for the last data definition language (DDL) create statement run by the user.                                                                                                                                                                                                                                                                  |
| sessiontimeout | integer   | The maximum time in seconds that a session remains inactive or idle before timing out. 0 indicates that no timeout is set. For information about the cluster's idle or inactive timeout setting, see <a href="#">Quotas and limits in Amazon Redshift</a> in the <i>Amazon Redshift Management Guide</i> .                                                   |
| external_id    | text      | Unique identifier of the user in the third-party identity provider.                                                                                                                                                                                                                                                                                          |

## Sample queries

The following command retrieves user information from SVL\_USER\_INFO.

```
SELECT * FROM SVL_USER_INFO;
```

## SVL\_VACUUM\_PERCENTAGE

The SVL\_VACUUM\_PERCENTAGE view reports the percentage of data blocks allocated to a table after performing a vacuum. This percentage number shows how much disk space was reclaimed. See the [VACUUM \(p. 1015\)](#) command for more information about the vacuum utility.

SVL\_VACUUM\_PERCENTAGE is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

| Column name | Data type | Description                                                                                                         |
|-------------|-----------|---------------------------------------------------------------------------------------------------------------------|
| xid         | bigint    | Transaction ID for the vacuum statement.                                                                            |
| table_id    | integer   | Table ID for the vacuumed table.                                                                                    |
| percentage  | bigint    | Percentage of data blocks after a vacuum (relative to the number of blocks in the table before the vacuum was run). |

### Sample query

The following query displays the percentage for a specific operation on table 100238:

```
select * from svl_vacuum_percentage
where table_id=100238 and xid=2200;

xid | table_id | percentage
-----+-----+-----
1337 | 100238 | 60
(1 row)
```

After this vacuum operation, the table contained 60 percent of the original blocks.

# System catalog tables

## Topics

- [PG\\_ATTRIBUTE\\_INFO \(p. 1671\)](#)
- [PG\\_CLASS\\_INFO \(p. 1671\)](#)
- [PG\\_DATABASE\\_INFO \(p. 1672\)](#)
- [PG\\_DEFAULT\\_ACL \(p. 1673\)](#)
- [PG\\_EXTERNAL\\_SCHEMA \(p. 1674\)](#)
- [PG\\_LIBRARY \(p. 1675\)](#)
- [PG\\_PROC\\_INFO \(p. 1676\)](#)

- [PG\\_STATISTIC\\_INDICATOR \(p. 1676\)](#)
- [PG\\_TABLE\\_DEF \(p. 1677\)](#)
- [PG\\_USER\\_INFO \(p. 1679\)](#)
- [Querying the catalog tables \(p. 1680\)](#)

The system catalogs store schema metadata, such as information about tables and columns. System catalog tables have a PG prefix.

The standard PostgreSQL catalog tables are accessible to Amazon Redshift users. For more information about PostgreSQL system catalogs, see [PostgreSQL system tables](#)

## PG\_ATTRIBUTE\_INFO

PG\_ATTRIBUTE\_INFO is an Amazon Redshift system view built on the PostgreSQL catalog table PG\_ATTRIBUTE and the internal catalog table PG\_ATTRIBUTE\_ACL. PG\_ATTRIBUTE\_INFO includes details about columns of a table or view, including column access control lists, if any.

### Table columns

PG\_ATTRIBUTE\_INFO shows the following column in addition to the columns in PG\_ATTRIBUTE.

| Column name | Data type | Description                                                                                     |
|-------------|-----------|-------------------------------------------------------------------------------------------------|
| attacl      | aclitem[] | The column-level access privileges, if any, that have been granted specifically on this column. |

## PG\_CLASS\_INFO

PG\_CLASS\_INFO is an Amazon Redshift system view built on the PostgreSQL catalog tables PG\_CLASS and PG\_CLASS\_EXTENDED. PG\_CLASS\_INFO includes details about table creation time and the current distribution style. For more information, see [Working with data distribution styles \(p. 49\)](#).

PG\_CLASS\_INFO is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

### Table columns

PG\_CLASS\_INFO shows the following columns in addition to the columns in PG\_CLASS.

| Column name           | Data type | Description                                                                                                                                 |
|-----------------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------|
| relcreationtime       | timestamp | Time in UTC that the table was created.                                                                                                     |
| releffectivediststyle | integer   | The distribution style of a table or, if the table uses automatic distribution, the current distribution style assigned by Amazon Redshift. |

The RELEFFECTIVEDISTSTYLE column in PG\_CLASS\_INFO indicates the current distribution style for the table. If the table uses automatic distribution, RELEFFECTIVEDISTSTYLE is 10, 11, or 12, which indicates whether the effective distribution style is AUTO (ALL), AUTO (EVEN), or AUTO (KEY). If the table uses

automatic distribution, the distribution style might initially show AUTO (ALL), then change to AUTO (EVEN) when the table grows or AUTO (KEY) if a column is found to be useful as a distribution key.

The following table gives the distribution style for each value in RELEFFECTIVEDISTSTYLE column:

| RELEFFECTIVEDISTSTYLE | Current distribution style |
|-----------------------|----------------------------|
| 0                     | EVEN                       |
| 1                     | KEY                        |
| 8                     | ALL                        |
| 10                    | AUTO (ALL)                 |
| 11                    | AUTO (EVEN)                |
| 12                    | AUTO (KEY)                 |

## Example

The following query returns the current distribution style of tables in the catalog.

```
select reloid as tableid,trim(nspname) as schemaname,trim(relname) as
 tablename,reldiststyle,releffectivediststyle,
CASE WHEN "reldiststyle" = 0 THEN 'EVEN'::text
 WHEN "reldiststyle" = 1 THEN 'KEY'::text
 WHEN "reldiststyle" = 8 THEN 'ALL'::text
 WHEN "releffectivediststyle" = 10 THEN 'AUTO(ALL)'::text
 WHEN "releffectivediststyle" = 11 THEN 'AUTO(EVEN)'::text
 WHEN "releffectivediststyle" = 12 THEN 'AUTO(KEY)'::text ELSE '<<UNKNOWN>>'::text END
as diststyle,relcreationtime
from pg_class_info a left join pg_namespace b on a.relnamespace=b.oid;
```

| tableid | schemaname | tablename | reldiststyle | releffectivediststyle | diststyle  | relcreationtime             |
|---------|------------|-----------|--------------|-----------------------|------------|-----------------------------|
| 3638033 | public     | customer  | 0            | 0                     | EVEN       | 2019-06-13 15:02:50.6666718 |
| 3638037 | public     | sales     | 1            | 1                     | KEY        | 2019-06-13 15:03:29.595007  |
| 3638035 | public     | lineitem  | 8            | 8                     | ALL        | 2019-06-13 15:03:01.378538  |
| 3638039 | public     | product   | 9            | 10                    | AUTO(ALL)  | 2019-06-13 15:03:42.691611  |
| 3638041 | public     | shipping  | 9            | 11                    | AUTO(EVEN) | 2019-06-13 15:03:53.69192   |
| 3638043 | public     | support   | 9            | 12                    | AUTO(KEY)  | 2019-06-13 15:03:59.120695  |

(6 rows)

## PG\_DATABASE\_INFO

PG\_DATABASE\_INFO is an Amazon Redshift system view that extends the PostgreSQL catalog table PG\_DATABASE.

PG\_DATABASE\_INFO is visible to all users.

## Table columns

PG\_DATABASE\_INFO contains the following columns in addition to columns in PG\_DATABASE. For more information, see the [PostgreSQL documentation](#).

| Column name  | Data type | Description                                                                                                   |
|--------------|-----------|---------------------------------------------------------------------------------------------------------------|
| datid        | oid       | The object identifier (OID) used internally by system tables.                                                 |
| datconnlimit | text      | The maximum number of concurrent connections that can be made to this database. A value of -1 means no limit. |

## PG\_DEFAULT\_ACL

Stores information about default access privileges. For more information on default access privileges, see [ALTER DEFAULT PRIVILEGES \(p. 631\)](#).

PG\_DEFAULT\_ACL is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name       | Data type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| defacluser        | integer   | ID of the user to which the listed privileges are applied.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| defaclnamespacoid |           | The object ID of the schema where default privileges are applied. The default value is 0 if no schema is specified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| defaclobjtype     | character | The type of object to which default privileges are applied. Valid values are as follows: <ul style="list-style-type: none"> <li>• r—relation (table or view)</li> <li>• f—function</li> <li>• p—stored procedure</li> </ul>                                                                                                                                                                                                                                                                                                                                                                            |
| defaclacl         | aclitem[] | A string that defines the default privileges for the specified user or user group and object type.<br><br>If the privileges are granted to a user, the string is in the following form:<br><br>$\{ \textit{username}=\textit{privilegestring}/\textit{grantor} \}$<br><i>username</i><br><br>The name of the user to which privileges are granted. If <i>username</i> is omitted, the privileges are granted to PUBLIC.<br><br>If the privileges are granted to a user group, the string is in the following form:<br><br>$\{ \textit{"group groupname}=\textit{privilegestring}/\textit{grantor"} \}$ |

| Column name | Data type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|             |           | <p><i>privilegestring</i></p> <p>A string that specifies which privileges are granted.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li>• r—SELECT (read)</li> <li>• a—INSERT (append)</li> <li>• w—UPDATE (write)</li> <li>• d—DELETE</li> <li>• x—Grants the privilege to create a foreign key constraint ( REFERENCES).</li> <li>• X—EXECUTE</li> <li>• *—Indicates that the user receiving the preceding privilege can in turn grant the same privilege to others (WITH GRANT OPTION).</li> </ul> <p><i>grantor</i></p> <p>The name of the user that granted the privileges.</p> <p>The following example indicates that the user admin granted all privileges, including WITH GRANT OPTION, to the user dbuser.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>dbuser=r*a*w*d*x*X*/admin</pre> </div> |

## Example

The following query returns all default privileges defined for the database.

```
select pg_get_userbyid(d.defacluser) as user,
n.nspname as schema,
case d.defaclobjtype when 'r' then 'tables' when 'f' then 'functions' end
as object_type,
array_to_string(d.defaclacl, ' + ') as default_privileges
from pg_catalog.pg_default_acl d
left join pg_catalog.pg_namespace n on n.oid = d.defaclnamespace;

user | schema | object_type | default_privileges
-----+-----+-----+-----
admin | tickit | tables | user1=r/admin + "group group1=a/admin" + user2=w/admin
```

The result in the preceding example shows that for all new tables created by user admin in the tickit schema, admin grants SELECT privileges to user1, INSERT privileges to group1, and UPDATE privileges to user2.

## PG\_EXTERNAL\_SCHEMA

Stores information about external schemas.

PG\_EXTERNAL\_SCHEMA is visible to all users. Superusers can see all rows; regular users can see only metadata to which they have access. For more information, see [CREATE EXTERNAL SCHEMA \(p. 762\)](#).

## Table columns

| Column name | Data type | Description              |
|-------------|-----------|--------------------------|
| esoid       | oid       | External schema ID.      |
| eskind      | integer   | Kind of external schema. |
| esdbname    | text      | External database name.  |
| esoptions   | text      | External schema options. |

## Example

The following example shows details for external schemas.

```
select esoid, nsname as schemaname, nspowner, esdbname as external_db, esoptions
from pg_namespace a,pg_external_schema b where a.oid=b.esoid;

esoid | schemaname | nspowner | external_db | esoptions
-----+-----+-----+-----+
+-----+
100134 | spectrum_schema | 100 | spectrum_db | [{"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}]
100135 | spectrum | 100 | spectrumbd | [{"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}]
100149 | external | 100 | external_db | [{"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}]
```

## PG\_LIBRARY

Stores information about user-defined libraries.

PG\_LIBRARY is visible to all users. Superusers can see all rows; regular users can see only their own data. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name   | Data type | Description                   |
|---------------|-----------|-------------------------------|
| name          | name      | Library name.                 |
| language_oid  | oid       | Reserved for system use.      |
| file_store_id | integer   | Reserved for system use.      |
| owner         | integer   | User ID of the library owner. |

## Example

The following example returns information for user-installed libraries.

```
select * from pg_library;
name | language_oid | file_store_id | owner
-----+-----+-----+-----
f_urlparse | 108254 | 2000 | 100
```

## PG\_PROC\_INFO

PG\_PROC\_INFO is an Amazon Redshift system view built on the PostgreSQL catalog table PG\_PROC and the internal catalog table PG\_PROC\_EXTENDED. PG\_PROC\_INFO includes details about stored procedures and functions, including information related to output arguments, if any.

### Table columns

PG\_PROC\_INFO shows the following columns in addition to the columns in PG\_PROC.

| Column name    | Data type | Description                                                                                                                                                                                                                                                                                |
|----------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| prooid         | oid       | The object ID of the function or stored procedure.                                                                                                                                                                                                                                         |
| prokind        | "char"    | A value that indicates the type of functions or stored procedures. This value is 'f' for regular functions, 'p' for stored procedures, and 'a' for aggregate functions.                                                                                                                    |
| proargmodes    | "char"[ ] | An array with the modes of the procedure arguments, encoded as 'i' for IN arguments, 'o' for OUT arguments, and 'b' for INOUT arguments. If all the arguments are IN arguments, this field is NULL. Subscripts correspond to positions in the proallargtypes array.                        |
| proallargtypes | oid[ ]    | An array with the data types of the procedure arguments. This array includes all types of arguments (including OUT and INOUT arguments). However, if all the arguments are IN arguments, this field is NULL. Subscripting is one-based. In contrast, proargtypes is subscripted from zero. |

The field proargnames in PG\_PROC\_INFO contains the names of all types of arguments (including OUT and INOUT), if any.

## PG\_STATISTIC\_INDICATOR

Stores information about the number of rows inserted or deleted since the last ANALYZE. The PG\_STATISTIC\_INDICATOR table is updated frequently following DML operations, so statistics are approximate.

PG\_STATISTIC\_INDICATOR is visible only to superusers. For more information, see [Visibility of data in system tables and views \(p. 1410\)](#).

## Table columns

| Column name | Data type | Description                                               |
|-------------|-----------|-----------------------------------------------------------|
| stairelid   | oid       | Table ID                                                  |
| stairows    | float     | Total number of rows in the table.                        |
| staiins     | float     | Number of rows inserted since the last ANALYZE.           |
| staidel     | float     | Number of rows deleted or updated since the last ANALYZE. |

## Example

The following example returns information for table changes since the last ANALYZE.

```
select * from pg_statistic_indicator;
stairelid | stairows | staiins | staidel
-----+-----+-----+
108271 | 11 | 0 | 0
108275 | 365 | 0 | 0
108278 | 8798 | 0 | 0
108280 | 91865 | 0 | 100632
108267 | 89981 | 49990 | 9999
108269 | 808 | 606 | 374
108282 | 152220 | 76110 | 248566
```

## PG\_TABLE\_DEF

Stores information about table columns.

PG\_TABLE\_DEF only returns information about tables that are visible to the user. If PG\_TABLE\_DEF does not return the expected results, verify that the [search\\_path \(p. 1696\)](#) parameter is set correctly to include the relevant schemas.

You can use [SVV\\_TABLE\\_INFO \(p. 1457\)](#) to view more comprehensive information about a table, including data distribution skew, key distribution skew, table size, and statistics.

## Table columns

| Column name | Data type     | Description         |
|-------------|---------------|---------------------|
| schemaname  | name          | Schema name.        |
| tablename   | name          | Table name.         |
| column      | name          | Column name.        |
| type        | text          | Datatype of column. |
| encoding    | character(32) | Encoding of column. |

| Column name | Data type | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| distkey     | boolean   | True if this column is the distribution key for the table.                                                                                                                                                                                                                                                                                                                                                                                                                            |
| sortkey     | integer   | Order of the column in the sort key. If the table uses a compound sort key, then all columns that are part of the sort key have a positive value that indicates the position of the column in the sort key. If the table uses an interleaved sort key, then each column that is part of the sort key has a value that is alternately positive or negative, where the absolute value indicates the position of the column in the sort key. If 0, the column is not part of a sort key. |
| notnull     | boolean   | True if the column has a NOT NULL constraint.                                                                                                                                                                                                                                                                                                                                                                                                                                         |

## Example

The following example shows the compound sort key columns for the LINEORDER\_COMPOUND table.

```
select "column", type, encoding, distkey, sortkey, "notnull"
from pg_table_def
where tablename = 'lineorder_compound'
and sortkey >= 0;

column | type | encoding | distkey | sortkey | notnull
-----+-----+-----+-----+-----+-----+
lo_orderkey | integer | delta32k | false | 1 | true
lo_custkey | integer | none | false | 2 | true
lo_partkey | integer | none | true | 3 | true
lo_suppkey | integer | delta32k | false | 4 | true
lo_orderdate| integer | delta | false | 5 | true
(5 rows)
```

The following example shows the interleaved sort key columns for the LINEORDER\_INTERLEAVED table.

```
select "column", type, encoding, distkey, sortkey, "notnull"
from pg_table_def
where tablename = 'lineorder_interleaved'
and sortkey >= 0;

column | type | encoding | distkey | sortkey | notnull
-----+-----+-----+-----+-----+-----+
lo_orderkey | integer | delta32k | false | -1 | true
lo_custkey | integer | none | false | 2 | true
lo_partkey | integer | none | true | -3 | true
lo_suppkey | integer | delta32k | false | 4 | true
lo_orderdate| integer | delta | false | -5 | true
(5 rows)
```

PG\_TABLE\_DEF will only return information for tables in schemas that are included in the search path. For more information, see [search\\_path \(p. 1696\)](#).

For example, suppose you create a new schema and a new table, then query PG\_TABLE\_DEF.

```
create schema demo;
create table demo.demotable (one int);
select * from pg_table_def where tablename = 'demotable';
```

| schemaname tablename column  type   encoding   distkey   sortkey   notnull |
|----------------------------------------------------------------------------|
|----------------------------------------------------------------------------|

The query returns no rows for the new table. Examine the setting for `search_path`.

```
show search_path;
-----+
$user, public
(1 row)
```

Add the demo schema to the search path and run the query again.

```
set search_path to '$user', 'public', 'demo';
select * from pg_table_def where tablename = 'demotable';
-----+
schemaname| tablename | column| type | encoding |distkey|sortkey| notnull
-----+-----+-----+-----+-----+-----+-----+
demo | demotable | one | integer | none | f | 0 | f
(1 row)
```

## PG\_USER\_INFO

`PG_USER_INFO` is an Amazon Redshift system view that shows user information, such as user ID and password expiration time.

Only superusers can see `PG_USER_INFO`.

### Table columns

`PG_USER_INFO` contains the following columns. For more information, see the [PostgreSQL documentation](#).

| Column name  | Data type | Description                                       |
|--------------|-----------|---------------------------------------------------|
| username     | name      | The username.                                     |
| usesysid     | integer   | The user ID.                                      |
| usecreatedb  | boolean   | True if the user can create databases.            |
| usesuper     | boolean   | True if the user is a superuser.                  |
| usecatupd    | boolean   | True if the user can update system catalogs.      |
| passwd       | text      | The password.                                     |
| valuntil     | abstime   | The password's expiration date and time.          |
| useconfig    | text[]    | The session defaults for run-time variables.      |
| useconnlimit | text      | The number of connections that the user can open. |

# Querying the catalog tables

## Topics

- [Examples of catalog queries \(p. 1681\)](#)

In general, you can join catalog tables and views (relations whose names begin with **PG\_**) to Amazon Redshift tables and views.

The catalog tables use a number of data types that Amazon Redshift does not support. The following data types are supported when queries join catalog tables to Amazon Redshift tables:

- `bool`
- `"char"`
- `float4`
- `int2`
- `int4`
- `int8`
- `name`
- `oid`
- `text`
- `varchar`

If you write a join query that explicitly or implicitly references a column that has an unsupported data type, the query returns an error. The SQL functions that are used in some of the catalog tables are also unsupported, except for those used by the `PG_SETTINGS` and `PG_LOCKS` tables.

For example, the `PG_STATS` table cannot be queried in a join with an Amazon Redshift table because of unsupported functions.

The following catalog tables and views provide useful information that can be joined to information in Amazon Redshift tables. Some of these tables allow only partial access because of data type and function restrictions. When you query the partially accessible tables, select or reference their columns carefully.

The following tables are completely accessible and contain no unsupported types or functions:

- [`pg\_attribute`](#)
- [`pg\_cast`](#)
- [`pg\_depend`](#)
- [`pg\_description`](#)
- [`pg\_locks`](#)
- [`pg\_opclass`](#)

The following tables are partially accessible and contain some unsupported types, functions, and truncated text columns. Values in text columns are truncated to `varchar(256)` values.

- [`pg\_class`](#)
- [`pg\_constraint`](#)
- [`pg\_database`](#)
- [`pg\_group`](#)
- [`pg\_language`](#)

- [pg\\_namespace](#)
- [pg\\_operator](#)
- [pg\\_proc](#)
- [pg\\_settings](#)
- [pg\\_statistic](#)
- [pg\\_tables](#)
- [pg\\_type](#)
- [pg\\_user](#)
- [pg\\_views](#)

The catalog tables that are not listed here are either inaccessible or unlikely to be useful to Amazon Redshift administrators. However, you can query any catalog table or view openly if your query does not involve a join to an Amazon Redshift table.

You can use the OID columns in the Postgres catalog tables as joining columns. For example, the join condition pg\_database.oid = stv\_tbl\_perm.db\_id matches the internal database object ID for each PG\_DATABASE row with the visible DB\_ID column in the STV\_TBL\_PERM table. The OID columns are internal primary keys that are not visible when you select from the table. The catalog views do not have OID columns.

Some Amazon Redshift functions must run only on the compute nodes. If a query references a user-created table, the SQL runs on the compute nodes.

A query that references only catalog tables (tables with a PG prefix, such as PG\_TABLE\_DEF) or that does not reference any tables, runs exclusively on the leader node.

If a query that uses a compute-node function doesn't reference a user-defined table or Amazon Redshift system table returns the following error.

[Amazon](500310) Invalid operation: One or more of the used functions must be applied on at least one user created table.

The following Amazon Redshift functions are compute-node only functions:

#### System information functions

- LISTAGG
- MEDIAN
- PERCENTILE\_CONT
- PERCENTILE\_DISC and APPROXIMATE\_PERCENTILE\_DISC

## Examples of catalog queries

The following queries show a few of the ways in which you can query the catalog tables to get useful information about an Amazon Redshift database.

### View table ID, database, schema, and table name

The following view definition joins the STV\_TBL\_PERM system table with the PG\_CLASS, PG\_NAMESPACE, and PG\_DATABASE system catalog tables to return the table ID, database name, schema name, and table name.

```
create view tables_vw as
```

```

select distinct(id) table_id
,trim(datname) db_name
,trim(nspname) schema_name
,trim(relname) table_name
from stv_tbl_perm
join pg_class on pg_class.oid = stv_tbl_perm.id
join pg_namespace on pg_namespace.oid = relnamespace
join pg_database on pg_database.oid = stv_tbl_perm.db_id;

```

The following example returns the information for table ID 117855.

```
select * from tables_vw where table_id = 117855;
```

| table_id | db_name | schema_name | table_name |
|----------|---------|-------------|------------|
| 117855   | dev     | public      | customer   |

## List the number of columns per Amazon Redshift table

The following query joins some catalog tables to find out how many columns each Amazon Redshift table contains. Amazon Redshift table names are stored in both PG\_TABLES and STV\_TBL\_PERM; where possible, use PG\_TABLES to return Amazon Redshift table names.

This query does not involve any Amazon Redshift tables.

```

select nspname, relname, max(attnum) as num_cols
from pg_attribute a, pg_namespace n, pg_class c
where n.oid = c.relnamespace and a.attrelid = c.oid
and c.relname not like '%pkey'
and n.nspname not like 'pg%'
and n.nspname not like 'information%'
group by 1, 2
order by 1, 2;

nspname | relname | num_cols
-----+-----+
public | category | 4
public | date | 8
public | event | 6
public | listing | 8
public | sales | 10
public | users | 18
public | venue | 5
(7 rows)

```

## List the schemas and tables in a database

The following query joins STV\_TBL\_PERM to some PG tables to return a list of tables in the TICKIT database and their schema names (NSPNAME column). The query also returns the total number of rows in each table. (This query is helpful when multiple schemas in your system have the same table names.)

```

select datname, nspname, relname, sum(rows) as rows
from pg_class, pg_namespace, pg_database, stv_tbl_perm
where pg_namespace.oid = relnamespace
and pg_class.oid = stv_tbl_perm.id
and pg_database.oid = stv_tbl_perm.db_id
and datname ='tickit'
group by datname, nspname, relname

```

```

order by datname, nspname, relname;

datname | nspname | relname | rows
-----+-----+-----+-----
ticket | public | category | 11
ticket | public | date | 365
ticket | public | event | 8798
ticket | public | listing | 192497
ticket | public | sales | 172456
ticket | public | users | 49990
ticket | public | venue | 202
(7 rows)

```

## List table IDs, data types, column names, and table names

The following query lists some information about each user table and its columns: the table ID, the table name, its column names, and the data type of each column:

```

select distinct attrelid, rtrim(name), attname, typename
from pg_attribute a, pg_type t, stv_tbl_perm p
where t.oid=a.atttypid and a.attrelid=p.id
and a.attrelid between 100100 and 110000
and typename not in('oid','xid','tid','cid')
order by a.attrelid asc, typename, attname;

attrelid | rtrim | attname | typename
-----+-----+-----+-----
100133 | users | likebroadway | bool
100133 | users | likeclassical | bool
100133 | users | likeconcerts | bool
...
100137 | venue | venuestate | bpchar
100137 | venue | venueid | int2
100137 | venue | venueseats | int4
100137 | venue | venuecity | varchar
...

```

## Count the number of data blocks for each column in a table

The following query joins the STV\_BLOCKLIST table to PG\_CLASS to return storage information for the columns in the SALES table.

```

select col, count(*)
from stv_blocklist s, pg_class p
where s.tbl=p.oid and relname='sales'
group by col
order by col;

col | count
----+-----
0 | 4
1 | 4
2 | 4
3 | 4
4 | 4
5 | 4
6 | 4
7 | 4
8 | 4
9 | 8
10 | 4
12 | 4

```

|           |  |   |
|-----------|--|---|
| 13        |  | 8 |
| (13 rows) |  |   |

# Configuration reference

## Modifying the server configuration

You can change the server configuration in the following ways:

- By using a [SET \(p. 977\)](#) command to override a setting for the duration of the current session only.

For example:

```
set extra_float_digits to 2;
```

- By modifying the parameter group settings for the cluster. The parameter group settings include additional parameters that you can configure. For more information, see [Amazon Redshift Parameter Groups](#) in the *Amazon Redshift Management Guide*.
- By using the [ALTER USER \(p. 659\)](#) command to set a configuration parameter to a new value for all sessions run by the specified user.

```
ALTER USER username SET parameter { TO | = } { value | DEFAULT }
```

Use the SHOW command to view the current parameter settings. Use SHOW ALL to view all the settings that you can configure by using the [SET \(p. 977\)](#) command.

```
show all;
```

| name                      | setting        |
|---------------------------|----------------|
| analyze_threshold_percent | 10             |
| datestyle                 | ISO, MDY       |
| extra_float_digits        | 2              |
| query_group               | default        |
| search_path               | \$user, public |
| statement_timeout         | 0              |
| timezone                  | UTC            |
| wlm_query_slot_count      | 1              |

## analyze\_threshold\_percent

### Values (default in bold)

**10**, 0 to 100.0

### Description

Sets the threshold for percentage of rows changed for analyzing a table. To reduce processing time and improve overall system performance, Amazon Redshift skips ANALYZE for any table that has a lower percentage of changed rows than specified by analyze\_threshold\_percent. For example, if a table

contains 100,000,000 rows and 9,000,000 rows have changed since the last ANALYZE, then by default the table is skipped because fewer than 10 percent of the rows have changed. To analyze tables when only a small number of rows have changed, set `analyze_threshold_percent` to an arbitrarily small number. For example, if you set `analyze_threshold_percent` to 0.01, then a table with 100,000,000 rows will not be skipped if at least 10,000 rows have changed. To analyze all tables even if no rows have changed, set `analyze_threshold_percent` to 0.

You can modify the `analyze_threshold_percent` parameter for the current session only by using a SET command. The parameter can't be modified in a parameter group.

## Example

```
set analyze_threshold_percent to 15;
set analyze_threshold_percent to 0.01;
set analyze_threshold_percent to 0;
```

## cast\_super\_null\_on\_error

### Values (default in bold)

on, off

### Description

Specifies that when you try to access a nonexistent member of an object or element of an array, Amazon Redshift returns a NULL value if your query is run in the default lax mode.

## datashare\_break\_glass\_session\_var

### Values (default in bold)

There is no default. The value can be any character string generated by Amazon Redshift when an operation occurs that isn't recommended, as described following.

### Description

Applies a permission that allows certain operations that generally aren't recommended for an AWS Data Exchange datashare.

In general, we recommend that you don't drop or alter an AWS Data Exchange datashare using the DROP DATASHARE or ALTER DATASHARE SET PUBLICACCESSIBLE statement. To allow dropping or altering an AWS Data Exchange datashare to turn off the publicly accessible setting, set the `datashare_break_glass_session_var` variable to a one-time value. This one-time value is generated by Amazon Redshift and provided in an error message after the initial attempt at the operation in question.

After setting the variable to the one-time generated value, run the DROP DATASHARE or ALTER DATASHARE statement again.

For more information, see [ALTER DATASHARE usage notes \(p. 630\)](#) or [DROP DATASHARE usage notes \(p. 864\)](#).

## Example

```
set datashare_break_glass_session_var to '620c871f890c49';
```

# datestyle

## Values (default in bold)

Format specification (**ISO**, Postgres, SQL, or German), and year/month/day ordering (**DMY**, **MDY**, **YMD**).

**ISO, MDY**

## Description

Sets the display format for date and time values and also the rules for interpreting ambiguous date input values. The string contains two parameters that you can change separately or together.

## Example

```
show datestyle;
DateStyle

ISO, MDY
(1 row)

set datestyle to 'SQL,DMY';
```

# default\_geometry\_encoding

## Values (default in bold)

1, 2

## Description

A session configuration that specifies if spatial geometries created during this session are encoded with a bounding box. If `default_geometry_encoding` is 1, then geometries are not encoded with a bounding box. If `default_geometry_encoding` is 2, then geometries are encoded with a bounding box. For more information about support for bounding boxes, see [Bounding box \(p. 229\)](#).

# describe\_field\_name\_in\_uppercase

## Values (default in bold)

**off (false)**, on (true)

## Description

Specifies whether column names returned by SELECT statements are uppercase or lowercase. If this parameter is on, column names are returned in uppercase. If this parameter is off, column names are returned in lowercase. Amazon Redshift stores column names in lowercase regardless of the setting for `describe_field_name_in_uppercase`.

## Example

```
set describe_field_name_in_uppercase to on;
show describe_field_name_in_uppercase;
DESCRIBE_FIELD_NAME_IN_UPPERCASE

on
```

## downcase\_delimited\_identifier

### Values (default in bold)

**on, off**

## Description

This configuration is being retired. Instead use `enable_case_sensitive_identifier`.

Enables the super parser to read JSON fields that are in uppercase or mixed case. Also enables federated query support to supported PostgreSQL databases with mixed-case names of database, schema, table, and column. To use case-sensitive identifiers, set this parameter to off.

## enable\_case\_sensitive\_identifier

### Values (default in bold)

**true, false**

## Description

A configuration value that determines whether name identifiers of databases, tables, and columns are case sensitive. The case of name identifiers is preserved when enclosed within double quotation marks. When you set `enable_case_sensitive_identifier` to `true`, the case of name identifiers is preserved. When you set `enable_case_sensitive_identifier` to `false`, the case of name identifiers is not preserved.

The case of a `username` enclosed in double quotation marks is always preserved regardless of the setting of the `enable_case_sensitive_identifier` configuration option.

## Examples

The following example shows how to create and use case sensitive identifiers for a table and column name.

```
-- To create and use case sensitive identifiers
SET enable_case_sensitive_identifier TO true;

-- Create tables and columns with case sensitive identifiers
CREATE TABLE "MixedCasedTable" ("MixedCasedColumn" int);

CREATE TABLE MixedCasedTable (MixedCasedColumn int);

-- Now query with case sensitive identifiers
SELECT "MixedCasedColumn" FROM "MixedCasedTable";

MixedCasedColumn

(0 rows)

SELECT MixedCasedColumn FROM MixedCasedTable;

mixedcasedcolumn

(0 rows)
```

The following example shows when the case of identifiers is not preserved.

```
-- To not use case sensitive identifiers
RESET enable_case_sensitive_identifier;

-- Mixed case identifiers are lowercased
CREATE TABLE "MixedCasedTable2" ("MixedCasedColumn" int);

CREATE TABLE MixedCasedTable2 (MixedCasedColumn int);

ERROR: Relation "mixedcasedtable2" already exists

SELECT "MixedCasedColumn" FROM "MixedCasedTable2";

mixedcasedcolumn

(0 rows)

SELECT MixedCasedColumn FROM MixedCasedTable2;

mixedcasedcolumn

(0 rows)
```

## Usage Notes

- If you're using autorefresh for materialized views, we recommend setting the `enable_case_sensitive_identifier` value in your cluster or workgroup's parameter group. This ensures that `enable_case_sensitive_identifier` stays constant when your materialized views are refreshed. For information on autorefresh for materialized views, see [Refreshing a materialized view \(p. 202\)](#). For information on setting configuration values in parameter groups, see [Amazon Redshift parameter groups](#) in the *Amazon Redshift Management Guide*.

## enable\_case\_sensitive\_super\_attribute

### Values (default in bold)

true, **false**

### Description

A configuration value that determines whether navigating SUPER data type structures is case sensitive. When you set `enable_case_sensitive_super_attribute` to `true`, navigating SUPER type structures is case sensitive. When you set the value to `false`, navigating SUPER type structures is not case sensitive.

When you enclose an attribute name in double quotation marks and set `enable_case_sensitive_identifier` to `true`, case is always preserved, regardless of the setting of the `enable_case_sensitive_super_attribute` configuration option.

`enable_case_sensitive_super_attribute` only applies to columns with the SUPER data type. For all other columns, consider using `enable_case_sensitive_identifier` instead.

For more information on the SUPER data type, see [SUPER type \(p. 594\)](#) and [Ingesting and querying semistructured data in Amazon Redshift \(p. 368\)](#).

### Examples

The following example shows the results of selecting SUPER values with `enable_case_sensitive_super_attribute` enabled and with it disabled.

```
--Create a table with a SUPER column.
CREATE TABLE tbl (col SUPER);

--Insert values.
INSERT INTO tbl VALUES (json_parse('{"
 "A": "A", "a": "a"
}'));

SET enable_case_sensitive_super_attribute TO ON;

SELECT col.A FROM tbl;
a

"A"
(1 row)

SELECT col.a FROM tbl;
```

```
a

"a"
(1 row)

SET enable_case_sensitive_super_attribute TO OFF;

SELECT col.A FROM tbl;
a

"a"
(1 row)

SELECT col.a FROM tbl;
a

"a"
(1 row)
```

## Usage Notes

- Views and materialized views follow the value of `enable_case_sensitive_super_attribute` at the time of their creation. Late-binding views, stored procedures, and user-defined functions follow the value of `enable_case_sensitive_super_attribute` at the time of querying.
- If you're using autorefresh for materialized views, we recommend setting the `enable_case_sensitive_identifier` value in your cluster or workgroup's parameter group. This ensures that `enable_case_sensitive_identifier` stays constant when your materialized views are refreshed. For information on autorefresh for materialized views, see [Refreshing a materialized view \(p. 202\)](#). For information on setting configuration values in parameter groups, see [Amazon Redshift parameter groups](#) in the *Amazon Redshift Management Guide*.
- The column name in statement results is always downcased, regardless of the value of `enable_case_sensitive_super_attribute`. To make the column name case sensitive as well, enable `enable_case_sensitive_identifier`.

## enable\_result\_cache\_for\_session

### Values (default in bold)

**on (true)**, off (false)

### Description

Specifies whether to use query results caching. If `enable_result_cache_for_session` is on, Amazon Redshift checks for a valid, cached copy of the query results when a query is submitted. If a match is found in the result cache, Amazon Redshift uses the cached results and doesn't run the query. If `enable_result_cache_for_session` is off, Amazon Redshift ignores the results cache and runs all queries when they are submitted.

### Example

```
SET enable_result_cache_for_session TO false;
--Amazon Redshift now ignores the results cache
```

## enable\_vacuum\_boost

### Values (default in bold)

false, true

### Description

Specifies whether to enable the vacuum boost option for all VACUUM commands run in a session. If enable\_vacuum\_boost is true, Amazon Redshift runs all VACUUM commands in the session with the BOOST option. If enable\_vacuum\_boost is false, Amazon Redshift doesn't run with the BOOST option by default. For more information about the BOOST option, see [VACUUM \(p. 1015\)](#).

## error\_on\_nondeterministic\_update

### Values (default in bold)

false, true

### Description

Specifies whether UPDATE queries with multiple matches per row return an error.

### Example

```
SET error_on_nondeterministic_update TO true;

CREATE TABLE t1(x1 int, y1 int);

CREATE TABLE t2(x2 int, y2 int);

INSERT INTO t1 VALUES (1,10), (2,20), (3,30);

INSERT INTO t2 VALUES (2,40), (2,50);

UPDATE t1 SET y1=y2 FROM t2 WHERE x1=x2;

ERROR: Found multiple matches to update the same tuple.
```

## extra\_float\_digits

### Values (default in bold)

0, -15 to 2

### Description

Sets the number of digits displayed for floating-point values, including float4 and float8. The value is added to the standard number of digits (FLT\_DIG or DBL\_DIG as appropriate). The value can be set as

high as 2, to include partially significant digits. This is especially useful for outputting float data that must be restored exactly. Or it can be set negative to suppress unwanted digits.

## json\_serialization\_enable

### Values (default in bold)

**false**, true

### Description

A session configuration that modifies the JSON serialization behavior of ORC, JSON, Ion, and Parquet formatted data. If `json_serialization_enable` is `true`, all top-level collections are automatically serialized to JSON and returned as `VARCHAR(65535)`. Noncomplex columns are not affected or serialized. Because collection columns are serialized as `VARCHAR(65535)`, their nested subfields can no longer be accessed directly as part of the query syntax (that is, in the filter clause). If `json_serialization_enable` is `false`, top-level collections are not serialized to JSON. For more information about nested JSON serialization, see [Serializing complex nested JSON \(p. 302\)](#).

## json\_serialization\_parse\_nested\_strings

### Values (default in bold)

**false**, true

### Description

A session configuration that modifies the JSON serialization behavior of ORC, JSON, Ion, and Parquet formatted data. When both `json_serialization_parse_nested_strings` and `json_serialization_enable` are `true`, string values that are stored in complex types (such as, maps, structures, or arrays) are parsed and written inline directly into the result if they are valid JSON. If `json_serialization_parse_nested_strings` is `false`, strings within nested complex types are serialized as escaped JSON strings. For more information, see [Serializing complex types containing JSON strings \(p. 303\)](#).

## max\_concurrency\_scaling\_clusters

### Values (default in bold)

1, 0 to 10

### Description

Sets the maximum number of concurrency scaling clusters allowed when concurrency scaling is enabled. Increase this value if more concurrency scaling is required. Decrease this value to reduce the usage of concurrency scaling clusters and the resulting billing charges.

The maximum number of concurrency scaling clusters is an adjustable quota. For more information, see [Amazon Redshift quotas](#) in the *Amazon Redshift Management Guide*.

## max\_cursor\_result\_set\_size

### Values (default in bold)

**0 (defaults to maximum)** - 14400000 MB

### Description

The `max_cursor_result_set_size` parameter is no longer used. For more information about cursor result set size, see [Cursor constraints \(p. 856\)](#).

## mv\_enable\_aqmv\_for\_session

### Values (default in bold)

true, false

### Description

Specifies whether Amazon Redshift can perform automatic query rewriting of materialized views at the session level.

## navigate\_super\_null\_on\_error

### Values (default in bold)

on, off

### Description

Specifies that when you try to navigate a nonexistent member of an object or element of an array, Amazon Redshift returns a NULL value if your query is run in the default lax mode.

## parse\_super\_null\_on\_error

### Values (default in bold)

off, on

### Description

Specifies that when Amazon Redshift tries to parse a nonexistent member of an object or element of an array, Amazon Redshift returns a NULL value if your query is run in the strict mode.

## pg\_federation\_repeatable\_read

### Values (default in bold)

*true, false*

### Description

Specifies the federated query transaction isolation level for the results from the PostgreSQL database.

- When pg\_federation\_repeatable\_read is true, federated transactions are processed with REPEATABLE READ isolation level semantics. This is the default.
- When pg\_federation\_repeatable\_read is false, federated transactions are processed with READ COMMITTED isolation level semantics.

For more information, see the following:

- [Considerations when accessing federated data with Amazon Redshift \(p. 246\)](#).
- [Managing concurrent write operations \(p. 118\)](#).

### Examples

The following command sets pg\_federation\_repeatable\_read to on for a session. The show command shows the value of the set value.

```
set pg_federation_repeatable_read to on;
```

```
show pg_federation_repeatable_read;
pg_federation_repeatable_read

on
```

## query\_group

### Values (default in bold)

No default; the value can be any character string.

### Description

Applies a user-defined label to a group of queries that are run during the same session. This label is captured in the query logs. You can use it to constrain results from the STL\_QUERY and STV\_INFLIGHT tables and the SVL\_QLOG view. For example, you can apply a separate label to every query that you run to uniquely identify queries without having to look up their IDs.

This parameter doesn't exist in the server configuration file and must be set at runtime with a SET command. Although you can use a long character string as a label, the label is truncated to 30 characters in the LABEL column of the STL\_QUERY table and the SVL\_QLOG view (and to 15 characters in STV\_INFLIGHT).

In the following example, query\_group is set to **Monday**, then several queries are run with that label.

```
set query_group to 'Monday';
SET
select * from category limit 1;
...
...
select query, pid, substring, elapsed, label
from svl_qlog where label ='Monday'
order by query;

query | pid | substring | elapsed | label
-----+-----+-----+
789 | 6084 | select * from category limit 1; | 65468 | Monday
790 | 6084 | select query, trim(label) from ... | 1260327 | Monday
791 | 6084 | select * from svl_qlog where .. | 2293547 | Monday
792 | 6084 | select count(*) from bigsales; | 108235617 | Monday
...
```

## search\_path

### Values (default in bold)

**'\$user', public, schema\_names**

A comma-separated list of existing schema names. If **'\$user'** is present, then the schema having the same name as SESSION\_USER is substituted, otherwise it is ignored. If **public** is present and no schema with the name **public** exists, it is ignored.

### Description

Specifies the order in which schemas are searched when an object (such as a table or a function) is referenced by a simple name with no schema component:

- Search paths aren't supported with external schemas and external tables. External tables must be explicitly qualified by an external schema.
- When objects are created without a specific target schema, they are placed in the first schema listed in the search path. If the search path is empty, the system returns an error.
- When objects with identical names exist in different schemas, the one found first in the search path is used.
- An object that isn't in any of the schemas in the search path can only be referenced by specifying its containing schema with a qualified (dotted) name.
- The system catalog schema, pg\_catalog, is always searched. If it is mentioned in the path, it is searched in the specified order. If not, it is searched before any of the path items.
- The current session's temporary-table schema, pg\_temp\_nnn, is always searched if it exists. It can be explicitly listed in the path by using the alias pg\_temp. If it is not listed in the path, it is searched first (even before pg\_catalog). However, the temporary schema is only searched for relation names (tables, views). It is not searched for function names.

### Example

The following example creates the schema ENTERPRISE and sets the search\_path to the new schema.

```
create schema enterprise;
set search_path to enterprise;
show search_path;

search_path

enterprise
(1 row)
```

The following example adds the schema ENTERPRISE to the default search\_path.

```
set search_path to '$user', public, enterprise;
show search_path;

search_path

"$user", public, enterprise
(1 row)
```

The following example adds the table FRONTIER to the schema ENTERPRISE.

```
create table enterprise.frontier (c1 int);
```

When the table PUBLIC.FRONTIER is created in the same database, and the user does not specify the schema name in a query, PUBLIC.FRONTIER takes precedence over ENTERPRISE.FRONTIER.

```
create table public.frontier(c1 int);
insert into enterprise.frontier values(1);
select * from frontier;

frontier

(0 rows)

select * from enterprise.frontier;

c1

1
(1 row)
```

## spectrum\_enable\_pseudo\_columns

### Values (default in bold)

**true**, false

### Description

You can disable the creation of pseudocolumns for a session by setting the spectrum\_enable\_pseudo\_columns configuration parameter to false.

### Example

The following command disables the creation of pseudocolumns for a session.

```
set spectrum_enable_pseudo_columns to false;
```

## enable\_spectrum\_oid

### Values (default in bold)

true, false

### Description

You can also disable only the \$spectrum\_oid pseudocolumn by setting the enable\_spectrum\_oid configuration parameter to false.

### Example

The following command disables the \$spectrum\_oid pseudocolumn by setting the enable\_spectrum\_oid configuration parameter to false.

```
set enable_spectrum_oid to false;
```

## spectrum\_query\_maxerror

### Values (default in bold)

-1, integer

### Description

You can specify an integer to indicate the maximum number of errors to accept before canceling the query. A negative value turns off maximum error data handling. The results are logged in [SPECTRUM\\_SCAN\\_ERROR \(p. 1659\)](#).

### Example

The following example assumes ORC data that contains surplus characters and characters that are not valid. The column definition for my\_string specifies a length of 3 characters. Following is the sample data for this example:

```
my_string

abcdef
gh♦
ab
```

The following commands set the maximum number of errors to 1 and perform the query.

```
set spectrum_query_maxerror to 1;
```

```
SELECT my_string FROM orc_data;
```

The query stops and the results are logged in [SVL\\_SPECTRUM\\_SCAN\\_ERROR \(p. 1659\)](#).

## statement\_timeout

### Values (default in bold)

**0 (turns off limitation)**, x milliseconds

### Description

Stops any statement that takes over the specified number of milliseconds.

The statement\_timeout value is the maximum amount of time a query can run before Amazon Redshift terminates it. This time includes planning, queueing in workload management (WLM), and execution time. Compare this time to WLM timeout (max\_execution\_time) and a QMR (query\_execution\_time), which include only execution time.

If WLM timeout (max\_execution\_time) is also specified as part of a WLM configuration, the lower of statement\_timeout and max\_execution\_time is used. For more information, see [WLM timeout \(p. 502\)](#).

### Example

Because the following query takes longer than 1 millisecond, it times out and is canceled.

```
set statement_timeout = 1;

select * from listing where listid>5000;
ERROR: Query (150) canceled on user's request
```

## stored\_proc\_log\_min\_messages

### Values (default in bold)

**LOG**, INFO, NOTICE, WARNING, EXCEPTION

### Description

Specifies the minimum logging level of raised stored procedure messages. Messages at or above the specified level are logged. The default is LOG (all messages are logged). The order of log levels from highest to lowest is as follows:

1. EXCEPTION
2. WARNING
3. NOTICE
4. INFO
5. LOG

For example if you specify a value of NOTICE, then messages are only logged for NOTICE, WARNING, and EXCEPTION.

## timezone

### Values (default in bold)

**UTC**, time zone

### Syntax

```
SET timezone { TO | = } [time_zone | DEFAULT]
SET time zone [time_zone | DEFAULT]
```

### Description

Sets the time zone for the current session. The time zone can be the offset from Universal Coordinated Time (UTC) or a time zone name.

#### Note

You can't set the `timezone` configuration parameter by using a cluster parameter group. The time zone can be set only for the current session by using a `SET` command. To set the time zone for all sessions run by a specific database user, use the [ALTER USER \(p. 659\)](#) command. `ALTER USER ... SET TIMEZONE` changes the time zone for subsequent sessions, not for the current session.

When you set the time zone using the `SET timezone` (one word) command with either `T0` or `=`, you can specify `time_zone` as a time zone name, a POSIX-style format offset, or an ISO-8601 format offset, as shown following.

```
SET timezone { TO | = } time_zone
```

When you set the time zone using the `SET time zone` command *without* `T0` or `=`, you can specify `time_zone` using an `INTERVAL` and also a time zone name, a POSIX-style format offset, or an ISO-8601 format offset, as shown following.

```
SET time zone time_zone
```

### Time zone formats

Amazon Redshift supports the following time zone formats:

- Time zone name
- INTERVAL
- POSIX-style time zone specification
- ISO-8601 offset

Because time zone abbreviations, such as PST or PDT, are defined as a fixed offset from UTC and don't include daylight savings time rules, the `SET` command doesn't support time zone abbreviations.

For more details on time zone formats, see the following.

**Time zone name** – The full time zone name, such as America/New\_York. Full time zone names can include daylight savings rules.

The following are examples of time zone names:

- Etc/Greenwich
- America/New\_York
- CST6CDT
- GB

**Note**

Many time zone names, such as EST, MST, NZ, and UCT, are also abbreviations.

To view a list of valid time zone names, run the following command.

```
select pg_timezone_names();
```

**INTERVAL** – An offset from UTC. For example, PST is -8:00 or -8 hours.

The following are examples of INTERVAL time zone offsets:

- -8:00
- -8 hours
- 30 minutes

**POSIX-style format** – A time zone specification in the form *STDoffset* or *STDoffsetDST*, where *STD* is a time zone abbreviation, *offset* is the numeric offset in hours west from UTC, and *DST* is an optional daylight-savings zone abbreviation. Daylight savings time is assumed to be one hour ahead of the given offset.

POSIX-style time zone formats use positive offsets west of Greenwich, in contrast to the ISO-8601 convention, which uses positive offsets east of Greenwich.

The following are examples of POSIX-style time zones:

- PST8
- PST8PDT
- EST5
- EST5EDT

**Note**

Amazon Redshift doesn't validate POSIX-style time zone specifications, so it is possible to set the time zone to an invalid value. For example, the following command doesn't return an error, even though it sets the time zone to an invalid value.

```
set timezone to 'xxx36';
```

**ISO-8601 Offset** – The offset from UTC in the form  $\pm[hh]:[mm]$ .

The following are examples of ISO-8601 offsets:

- -8:00

- +7:30

## Examples

The following example sets the time zone for the current session to New York.

```
set timezone = 'America/New_York';
```

The following example sets the time zone for the current session to UTC–8 (PST).

```
set timezone to '-8:00';
```

The following example uses INTERVAL to set the time zone to PST.

```
set timezone interval '-8 hours'
```

The following example resets the time zone for the current session to the system default time zone (UTC).

```
set timezone to default;
```

To set the time zone for database user, use an ALTER USER ... SET statement. The following example sets the time zone for dbuser to New York. The new value persists for the user for all subsequent sessions.

```
ALTER USER dbuser SET timezone to 'America/New_York';
```

## wlm\_query\_slot\_count

### Values (default in bold)

1, 1 to 50 (cannot exceed number of available slots (concurrency level) for the service class)

### Description

Sets the number of query slots that a query uses.

Workload management (WLM) reserves slots in a service class according to the concurrency level set for the queue. For example, if concurrency level is set to 5, then the service class has 5 slots. WLM allocates the available memory for a service class equally to each slot. For more information, see [Implementing workload management \(p. 491\)](#).

#### Note

If the value of wlm\_query\_slot\_count is larger than the number of available slots (concurrency level) for the service class, the query fails. If you encounter an error, decrease wlm\_query\_slot\_count to an allowable value.

For operations where performance is heavily affected by the amount of memory allocated, such as vacuuming, increasing the value of wlm\_query\_slot\_count can improve performance. In particular, for slow vacuum commands, inspect the corresponding record in the SVV\_VACUUM\_SUMMARY view. If you see high values (close to or higher than 100) for sort\_partitions and merge\_increments in the

SVV\_VACUUM\_SUMMARY view, consider increasing the value for wlm\_query\_slot\_count the next time you run Vacuum against that table.

Increasing the value of wlm\_query\_slot\_count limits the number of concurrent queries that can be run. For example, suppose that the service class has a concurrency level of 5 and wlm\_query\_slot\_count is set to 3. While a query is running within the session with wlm\_query\_slot\_count set to 3, a maximum of 2 more concurrent queries can be run within the same service class. Subsequent queries wait in the queue until currently running queries complete and slots are freed.

## Examples

Use the SET command to set the value of wlm\_query\_slot\_count for the duration of the current session.

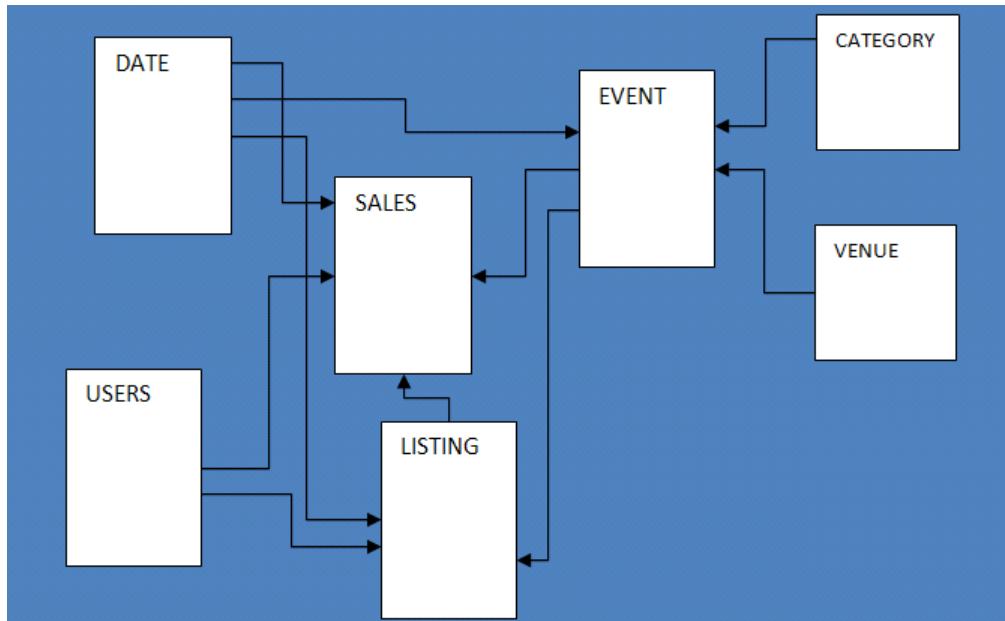
```
set wlm_query_slot_count to 3;
```

# Sample database

## Topics

- [CATEGORY table \(p. 1705\)](#)
- [DATE table \(p. 1705\)](#)
- [EVENT table \(p. 1706\)](#)
- [VENUE table \(p. 1706\)](#)
- [USERS table \(p. 1707\)](#)
- [LISTING table \(p. 1707\)](#)
- [SALES table \(p. 1708\)](#)

Most of the examples in the Amazon Redshift documentation use a sample database called TICKIT. This small database consists of seven tables: two fact tables and five dimensions. You can load the TICKIT dataset by following the steps in [Step 6: Load sample data from amazon S3](#) in the Amazon Redshift Getting Started Guide.



This sample database application helps analysts track sales activity for the fictional TICKIT web site, where users buy and sell tickets online for sporting events, shows, and concerts. In particular, analysts can identify ticket movement over time, success rates for sellers, and the best-selling events, venues, and seasons. Analysts can use this information to provide incentives to buyers and sellers who frequent the site, to attract new users, and to drive advertising and promotions.

For example, the following query finds the top five sellers in San Diego, based on the number of tickets sold in 2008:

```

select sellerid, username, (firstname || ' ' || lastname) as name,
city, sum(qtysold)
from sales, date, users

```

```

where sales.sellerid = users.userid
and sales.dateid = date.dateid
and year = 2008
and city = 'San Diego'
group by sellerid, username, name, city
order by 5 desc
limit 5;

sellerid | username | name | city | sum
-----+-----+-----+-----+-----
49977 | JJK84WTE | Julie Hanson | San Diego | 22
19750 | AAS23BDR | Charity Zimmerman | San Diego | 21
29069 | SVL81MEQ | Axel Grant | San Diego | 17
43632 | VAG08HKW | Griffin Dodson | San Diego | 16
36712 | RXT40MKU | Hiram Turner | San Diego | 14
(5 rows)

```

The database used for the examples in this guide contains a small data set; the two fact tables each contain less than 200,000 rows, and the dimensions range from 11 rows in the CATEGORY table up to about 50,000 rows in the USERS table.

In particular, the database examples in this guide demonstrate the key features of Amazon Redshift table design:

- Data distribution
- Data sort
- Columnar compression

## CATEGORY table

| Column name | Data type   | Description                                                                                                                      |
|-------------|-------------|----------------------------------------------------------------------------------------------------------------------------------|
| CATID       | SMALLINT    | Primary key, a unique ID value for each row. Each row represents a specific type of event for which tickets are bought and sold. |
| CATGROUP    | VARCHAR(10) | Descriptive name for a group of events, such as <b>Shows</b> and <b>Sports</b> .                                                 |
| CATNAME     | VARCHAR(10) | Short descriptive name for a type of event within a group, such as <b>Opera</b> and <b>Musicals</b> .                            |
| CATDESC     | VARCHAR(30) | Longer descriptive name for the type of event, such as <b>Musical theatre</b> .                                                  |

## DATE table

| Column name | Data type | Description                                                                                  |
|-------------|-----------|----------------------------------------------------------------------------------------------|
| DATEID      | SMALLINT  | Primary key, a unique ID value for each row. Each row represents a day in the calendar year. |
| CALDATE     | DATE      | Calendar date, such as <b>2008-06-24</b> .                                                   |

| Column name | Data type | Description                                                   |
|-------------|-----------|---------------------------------------------------------------|
| DAY         | CHAR(3)   | Day of week (short form), such as <b>SA</b> .                 |
| WEEK        | SMALLINT  | Week number, such as <b>26</b> .                              |
| MONTH       | CHAR(5)   | Month name (short form), such as <b>JUN</b> .                 |
| QTR         | CHAR(5)   | Quarter number ( <b>1</b> through <b>4</b> ).                 |
| YEAR        | SMALLINT  | Four-digit year ( <b>2008</b> ).                              |
| HOLIDAY     | BOOLEAN   | Flag that denotes whether the day is a public holiday (U.S.). |

## EVENT table

| Column name | Data type    | Description                                                                                                                                |
|-------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| EVENTID     | INTEGER      | Primary key, a unique ID value for each row. Each row represents a separate event that takes place at a specific venue at a specific time. |
| VENUEID     | SMALLINT     | Foreign-key reference to the VENUE table.                                                                                                  |
| CATID       | SMALLINT     | Foreign-key reference to the CATEGORY table.                                                                                               |
| DATEID      | SMALLINT     | Foreign-key reference to the DATE table.                                                                                                   |
| EVENTNAME   | VARCHAR(200) | Name of the event, such as <b>Hamlet</b> or <b>La Traviata</b> .                                                                           |
| STARTTIME   | TIMESTAMP    | Full date and start time for the event, such as <b>2008-10-10 19:30:00</b> .                                                               |

## VENUE table

| Column name | Data type    | Description                                                                                                                                                    |
|-------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VENUEID     | SMALLINT     | Primary key, a unique ID value for each row. Each row represents a specific venue where events take place.                                                     |
| VENUENAME   | VARCHAR(100) | Exact name of the venue, such as <b>Cleveland Browns Stadium</b> .                                                                                             |
| VENUECITY   | VARCHAR(30)  | City name, such as <b>Cleveland</b> .                                                                                                                          |
| VENUESTATE  | CHAR(2)      | Two-letter state or province abbreviation (United States and Canada), such as <b>OH</b> .                                                                      |
| VENUESEATS  | INTEGER      | Maximum number of seats available at the venue, if known, such as <b>73200</b> . For demonstration purposes, this column contains some null values and zeroes. |

## USERS table

| Column name     | Data type    | Description                                                                                                                                                             |
|-----------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| USERID          | INTEGER      | Primary key, a unique ID value for each row. Each row represents a registered user (a buyer or seller or both) who has listed or bought tickets for at least one event. |
| USERNAME        | CHAR(8)      | An 8-character alphanumeric username, such as <b>PGL08LJI</b> .                                                                                                         |
| FIRSTNAME       | VARCHAR(30)  | The user's first name, such as <b>Victor</b> .                                                                                                                          |
| LASTNAME        | VARCHAR(30)  | The user's last name, such as <b>Hernandez</b> .                                                                                                                        |
| CITY            | VARCHAR(30)  | The user's home city, such as <b>Naperville</b> .                                                                                                                       |
| STATE           | CHAR(2)      | The user's home state, such as <b>GA</b> .                                                                                                                              |
| EMAIL           | VARCHAR(100) | The user's email address; this column contains random Latin values, such as <b>turpis@accumsanlaoreet.org</b> .                                                         |
| PHONE           | CHAR(14)     | The user's 14-character phone number, such as <b>(818) 765-4255</b> .                                                                                                   |
| LIKESPORTS, ... | BOOLEAN      | A series of 10 different columns that identify the user's likes and dislikes with <b>true</b> and <b>false</b> values.                                                  |

## LISTING table

| Column name    | Data type    | Description                                                                                                            |
|----------------|--------------|------------------------------------------------------------------------------------------------------------------------|
| LISTID         | INTEGER      | Primary key, a unique ID value for each row. Each row represents a listing of a batch of tickets for a specific event. |
| SELLERID       | INTEGER      | Foreign-key reference to the USERS table, identifying the user who is selling the tickets.                             |
| EVENTID        | INTEGER      | Foreign-key reference to the EVENT table.                                                                              |
| DATEID         | SMALLINT     | Foreign-key reference to the DATE table.                                                                               |
| NUMTICKETS     | SMALLINT     | The number of tickets available for sale, such as <b>2</b> or <b>20</b> .                                              |
| PRICEPERTICKET | DECIMAL(8,2) | The fixed price of an individual ticket, such as <b>27.00</b> or <b>206.00</b> .                                       |
| TOTALPRICE     | DECIMAL(8,2) | The total price for this listing (NUMTICKETS*PRICEPERTICKET).                                                          |
| LISTTIME       | TIMESTAMP    | The full date and time when the listing was posted, such as <b>2008-03-18 07:19:35</b> .                               |

## SALES table

| Column name | Data type    | Description                                                                                                                                            |
|-------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| SALESID     | INTEGER      | Primary key, a unique ID value for each row. Each row represents a sale of one or more tickets for a specific event, as offered in a specific listing. |
| LISTID      | INTEGER      | Foreign-key reference to the LISTING table.                                                                                                            |
| SELLERID    | INTEGER      | Foreign-key reference to the USERS table (the user who sold the tickets).                                                                              |
| BUYERID     | INTEGER      | Foreign-key reference to the USERS table (the user who bought the tickets).                                                                            |
| EVENTID     | INTEGER      | Foreign-key reference to the EVENT table.                                                                                                              |
| DATEID      | SMALLINT     | Foreign-key reference to the DATE table.                                                                                                               |
| QTYSOLD     | SMALLINT     | The number of tickets that were sold, from <b>1</b> to <b>8</b> . (A maximum of 8 tickets can be sold in a single transaction.)                        |
| PRICEPAID   | DECIMAL(8,2) | The total price paid for the tickets, such as <b>75.00</b> or <b>488.00</b> . The individual price of a ticket is PRICEPAID/QTYSOLD.                   |
| COMMISSION  | DECIMAL(8,2) | The 15% commission that the business collects from the sale, such as <b>11.25</b> or <b>73.20</b> . The seller receives 85% of the PRICEPAID value.    |
| SALETIME    | TIMESTAMP    | The full date and time when the sale was completed, such as <b>2008-05-24 06:21:47</b> .                                                               |

# Appendix: Time zone names and abbreviations

The following topics describe how to find the valid time zone names and time zone abbreviations that can be specified with the [CONVERT\\_TIMEZONE function \(p. 1095\)](#).

## Topics

- [Time zone names \(p. 1709\)](#)
- [Time zone abbreviations \(p. 1709\)](#)

## Time zone names

The following SQL statement returns the valid time zone names that can be specified with the [CONVERT\\_TIMEZONE function \(p. 1095\)](#). For a current, complete list of time zone names, run the following command.

```
select pg_timezone_names();
```

Each row contains a comma-separated string with the time zone name, abbreviation, UTC offset, and indicator if the time zone observes daylight-savings (t or f). For example, the following snippet shows two resulting rows. The first row is the time zone Europe/Paris, abbreviation CET, with 01:00:00 offset from UTC, and f to indicate it does not observe daylight-savings time. The second row is the time zone Israel, abbreviation IST, with 02:00:00 offset from UTC, and f to indicate it does not observe daylight-savings time.

```
pg_timezone_names

(Europe/Paris,CET,01:00:00,f)
(Israel,IST,02:00:00,f)
```

Run the SQL statement to obtain the entire list and find a time zone name. Approximately 600 rows are returned. Even though some of the returned time zone names are capitalized initialisms or acronyms (for example; GB, PRC, ROK), the CONVERT\_TIMEZONE function treats them as time zone names, not time zone abbreviations.

## Time zone abbreviations

The following SQL statement returns the valid time zone abbreviations that can be specified with the [CONVERT\\_TIMEZONE function \(p. 1095\)](#).

```
select pg_timezone_abbrevs();
```

The results contain a comma-separated string with the time zone abbreviation, UTC offset, and indicator if the time zone observes daylight-savings (t or f). For example, the following snippet shows two resulting rows. The first row contains the abbreviation for Pacific Daylight Time PDT, with a -07:00:00

offset from UTC, and t to indicate it observes daylight-savings time. The second row contains the abbreviation for Pacific Standard Time PST, with a -08:00:00 offset from UTC, and f to indicate it does not observe daylight-savings time.

```
pg_timezone_abbrevs

(PDT,-07:00:00,t)
(PST,-08:00:00,f)
```

Run the SQL statement to obtain the entire list and find an abbreviation based on its offset and daylight-savings indicator. Approximately 200 rows are returned.

# Document history

The following table describes the important changes in each release of the *Amazon Redshift Database Developer Guide* after May 2018. For notification about updates to this documentation, you can subscribe to an RSS feed.

**API version: 2012-12-01**

**Latest documentation update: August 27, 2021**

For a list of the changes to the *Amazon Redshift Management Guide*, see [Amazon Redshift Management Guide Document History](#).

For more information about new features, including a list of fixes and the associated cluster version numbers for each release, see [Cluster Version History](#).

| Change                                                                                                                            | Description                                                                                                                                                                     | Date             |
|-----------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#"><u>Support for spatial 3D and 4D geometries and new spatial functions</u></a>                                         | You can now use additional spatial functions and 3D and 4D geometry support is added to some functions.                                                                         | August 19, 2021  |
| <a href="#"><u>Support for column compression encoding for automatic table optimization</u></a>                                   | You can specify the ENCODE AUTO option for a table to automatically manage compression encoding for all columns in the table.                                                   | August 3, 2021   |
| <a href="#"><u>Support for multiple SQL statements or an SQL statement with parameters using the Amazon Redshift Data API</u></a> | You can now run multiple SQL statements or a statement with parameters with the Amazon Redshift Data API.                                                                       | July 28, 2021    |
| <a href="#"><u>Support for case-insensitive collation with column level overrides</u></a>                                         | You can now use the COLLATE clause within a CREATE DATABASE statement to specify the default collation.                                                                         | June 24, 2021    |
| <a href="#"><u>Support for data sharing across accounts</u></a>                                                                   | You can now share data across AWS accounts.                                                                                                                                     | April 30, 2021   |
| <a href="#"><u>Support for hierarchical data queries with recursive CTE</u></a>                                                   | You can now use a recursive common table expression (CTE) in your SQL.                                                                                                          | April 29, 2021   |
| <a href="#"><u>Support for cross-database queries</u></a>                                                                         | You can now query data across databases in a cluster.                                                                                                                           | March 10, 2021   |
| <a href="#"><u>Support for fine-grained access control on COPY and UNLOAD commands</u></a>                                        | You can now grant the privilege to run COPY and UNLOAD commands to specific users and groups in your Amazon Redshift cluster to create more fine-grained access control policy. | January 12, 2021 |

|                                                                                              |                                                                                                                                                                                                 |                    |
|----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| <a href="#"><u>Support for native JSON and semi-structured data</u></a>                      | You can now define the SUPER data type.                                                                                                                                                         | December 9, 2020   |
| <a href="#"><u>Support for federated query to MySQL</u></a>                                  | You can now write a federated query to a supported MySQL engine.                                                                                                                                | December 9, 2020   |
| <a href="#"><u>Support for data sharing</u></a>                                              | You can now share data across Amazon Redshift clusters.                                                                                                                                         | December 9, 2020   |
| <a href="#"><u>Support for automatic table optimization</u></a>                              | You can now define automatic distribution and sort keys.                                                                                                                                        | December 9, 2020   |
| <a href="#"><u>Support for Amazon Redshift ML</u></a>                                        | You can now create, train, and deploy machine learning (ML) models.                                                                                                                             | December 8, 2020   |
| <a href="#"><u>Support for automatic refresh and query rewrite of materialized views</u></a> | You can now keep materialized views up-to-date with automatic refresh and query performance can be improved with automatic rewrite.                                                             | November 11, 2020  |
| <a href="#"><u>Support for TIME and TIMETZ data types</u></a>                                | You can now create tables with TIME and TIMETZ data types. TIME data type stores the time of day without timezone information, and TIMETZ stores the time of day including timezone information | November 11, 2020  |
| <a href="#"><u>Support for Lambda UDFs and tokenization</u></a>                              | You can now can write Lambda UDFs to enable external tokenization of data.                                                                                                                      | October 26, 2020   |
| <a href="#"><u>Support for altering a table column encoding</u></a>                          | You can now alter a table column encoding.                                                                                                                                                      | October 20, 2020   |
| <a href="#"><u>Support for querying across databases</u></a>                                 | Amazon Redshift can now query across databases in a cluster.                                                                                                                                    | October 15, 2020   |
| <a href="#"><u>Support for HyperLogLog Sketches</u></a>                                      | Amazon Redshift can now store and process HyperLogLogSketches.                                                                                                                                  | October 2, 2020    |
| <a href="#"><u>Support for Apache Hudi and Delta Lake</u></a>                                | Enhancements to creating external tables for Redshift Spectrum.                                                                                                                                 | September 24, 2020 |
| <a href="#"><u>Support for enhancements to querying spatial data</u></a>                     | Enhancements include loading a shapefile and several new spatial SQL functions.                                                                                                                 | September 15, 2020 |
| <a href="#"><u>Materialized view support external tables</u></a>                             | You can create materialized views in Amazon Redshift that reference external data sources.                                                                                                      | June 19, 2020      |

|                                                                            |                                                                                                                                                                             |                    |
|----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| <a href="#"><u>Support to write to external table</u></a>                  | You can write to external tables by running CREATE EXTERNAL TABLE AS SELECT to write to a new external table or INSERT INTO to insert data into an existing external table. | June 8, 2020       |
| <a href="#"><u>Support for storage controls for schemas</u></a>            | Updates to commands and views that manage storage controls for schemas.                                                                                                     | June 2, 2020       |
| <a href="#"><u>Support for federated query general availability</u></a>    | Updated information about querying data with federated queries.                                                                                                             | April 16, 2020     |
| <a href="#"><u>Support for additional spatial functions</u></a>            | Added descriptions of additional spatial functions.                                                                                                                         | April 2, 2020      |
| <a href="#"><u>Support for materialized views general availability</u></a> | Materialized views are generally available starting with cluster version 1.0.13059.                                                                                         | February 19, 2020  |
| <a href="#"><u>Support for column-level privileges</u></a>                 | Column-level privileges are available starting with cluster version 1.0.13059.                                                                                              | February 19, 2020  |
| <a href="#"><u>ALTER TABLE</u></a>                                         | You can use an ALTER TABLE command with the ALTER DISTSTYLE ALL clause to change the distribution style of a table.                                                         | February 11, 2020  |
| <a href="#"><u>Support for federated query</u></a>                         | Updated the guide to describe federated query with an updated CREATE EXTERNAL SCHEMA.                                                                                       | December 3, 2019   |
| <a href="#"><u>Support for data lake export</u></a>                        | Updated the guide to describe new parameters of the UNLOAD command.                                                                                                         | December 3, 2019   |
| <a href="#"><u>Support for spatial data</u></a>                            | Updated the guide to describe support for spatial data.                                                                                                                     | November 21, 2019  |
| <a href="#"><u>Support for the new console</u></a>                         | Updated the guide to describe the new Amazon Redshift console.                                                                                                              | November 11, 2019  |
| <a href="#"><u>Support for automatic table sort</u></a>                    | Amazon Redshift can automatically sort table data.                                                                                                                          | November 7, 2019   |
| <a href="#"><u>Support for VACUUM BOOST option</u></a>                     | You can use the BOOST option when vacuuming tables.                                                                                                                         | November 7, 2019   |
| <a href="#"><u>Support for default IDENTITY columns</u></a>                | You can create tables with default IDENTITY columns.                                                                                                                        | September 19, 2019 |
| <a href="#"><u>Support for AZ64 compression encoding</u></a>               | You can encode some columns with AZ64 compression encoding.                                                                                                                 | September 19, 2019 |

|                                                                                         |                                                                                                                                                                                                                                                                                                                                   |                  |
|-----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#"><u>Support for query priority</u></a>                                       | You can set the query priority of an automatic WLM queue.                                                                                                                                                                                                                                                                         | August 22, 2019  |
| <a href="#"><u>Support for AWS Lake Formation</u></a>                                   | You can use a Lake Formation Data Catalog with Amazon Redshift Spectrum.                                                                                                                                                                                                                                                          | August 8, 2019   |
| <a href="#"><u>COMPUPDATE PRESET</u></a>                                                | You can use a COPY command with COMPUPDATE PRESET to enable Amazon Redshift to choose the compression encoding.                                                                                                                                                                                                                   | June 13, 2019    |
| <a href="#"><u>ALTER COLUMN</u></a>                                                     | You can use an ALTER TABLE command with ALTER COLUMN to increase the size of a VARCHAR column.                                                                                                                                                                                                                                    | May 22, 2019     |
| <a href="#"><u>Support for stored procedures</u></a>                                    | You can define PL/pgSQL stored procedures in Amazon Redshift.                                                                                                                                                                                                                                                                     | April 24, 2019   |
| <a href="#"><u>Support for an automatic workload management (WLM) configuration</u></a> | You can enable Amazon Redshift to run with automatic WLM.                                                                                                                                                                                                                                                                         | April 24, 2019   |
| <a href="#"><u>UNLOAD to Zstandard</u></a>                                              | You can use the UNLOAD command to apply Zstandard compression to text and comma-separated value (CSV) files unloaded to Amazon S3.                                                                                                                                                                                                | April 3, 2019    |
| <a href="#"><u>Concurrency scaling</u></a>                                              | When concurrency scaling is enabled, Amazon Redshift automatically adds additional cluster capacity when you need it to process an increase in concurrent read queries.                                                                                                                                                           | March 21, 2019   |
| <a href="#"><u>UNLOAD to CSV</u></a>                                                    | You can use the UNLOAD command to unload to a file formatted as CSV text.                                                                                                                                                                                                                                                         | March 13, 2019   |
| <a href="#"><u>AUTO distribution style</u></a>                                          | To enable automatic distribution, you can specify the AUTO distribution style with a <a href="#"><u>CREATE TABLE</u></a> statement. When you enable automatic distribution, Amazon Redshift assigns an optimal distribution style based on the table data. The change in distribution occurs in the background, in a few seconds. | January 23, 2019 |

|                                                                              |                                                                                                                                                                                                                                                                                                   |                  |
|------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">COPY from Parquet supports SMALLINT (p. 1711)</a>                | COPY now supports loading from Parquet formatted files into columns that use the SMALLINT data type. For more information, see <a href="#">COPY from Columnar Data Formats</a>                                                                                                                    | January 2, 2019  |
| <a href="#">DROP EXTERNAL DATABASE</a>                                       | You can drop an external database by including the DROP EXTERNAL DATABASE clause with a <a href="#">DROP SCHEMA</a> command.                                                                                                                                                                      | December 3, 2018 |
| <a href="#">Cross-region UNLOAD</a>                                          | You can UNLOAD to an Amazon S3 bucket in another AWS Region by specifying the REGION parameter.                                                                                                                                                                                                   | October 31, 2018 |
| <a href="#">Automatic vacuum delete</a>                                      | Amazon Redshift automatically runs a <a href="#">VACUUM DELETE</a> operation in the background, so you rarely, if ever, need to run a DELETE ONLY vacuum. Amazon Redshift schedules the VACUUM DELETE to run during periods of reduced load and pauses the operation during periods of high load. | October 31, 2018 |
| <a href="#">Automatic distribution</a>                                       | When you don't specify a distribution style with a <a href="#">CREATE TABLE</a> statement, Amazon Redshift assigns an optimal distribution style based on the table data. The change in distribution occurs in the background, in a few seconds.                                                  | October 31, 2018 |
| <a href="#">Fine grained access control for the AWS Glue Data Catalog</a>    | You can now specify levels of access to data stored in the AWS Glue Data Catalog.                                                                                                                                                                                                                 | October 15, 2018 |
| <a href="#">UNLOAD with data types</a>                                       | You can specify the MANIFEST VERBOSE option with an <a href="#">UNLOAD</a> command to add metadata to the manifest file, including the names and data types of columns, file sizes, and row counts.                                                                                               | October 10, 2018 |
| <a href="#">Add multiple partitions using a single ALTER TABLE statement</a> | For Redshift Spectrum external tables, you can combine multiple PARTITION clauses in a single <a href="#">ALTER TABLE ADD</a> statement. For more information, see <a href="#">Alter External Table Examples</a> .                                                                                | October 10, 2018 |

|                                                                          |                                                                                                                                                                                                                                                                                  |                    |
|--------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| <a href="#">UNLOAD with header</a>                                       | You can specify the HEADER option with an <a href="#">UNLOAD</a> command to add a header line containing column names at the top of each output file.                                                                                                                            | September 19, 2018 |
| <a href="#">New system table and views</a>                               | <a href="#">SVL_S3Retries</a> , <a href="#">SVL_USER_INFO</a> , and <a href="#">STL_DISK_FULL_DIAG</a> documentation added.                                                                                                                                                      | August 31, 2018    |
| <a href="#">Support for nested data in Amazon Redshift Spectrum</a>      | You can now query nested data stored in Amazon Redshift Spectrum tables. For more information, see <a href="#">Tutorial: Querying Nested Data with Amazon Redshift Spectrum</a> .                                                                                                | August 8, 2018     |
| <a href="#">SQA on by default</a>                                        | Short query acceleration (SQA) is now enabled by default for all new clusters. SQA uses machine learning to provide higher performance, faster results, and better predictability of query execution times. For more information, see <a href="#">Short Query Acceleration</a> . | August 8, 2018     |
| <a href="#">Amazon Redshift Advisor</a>                                  | You can now get tailored recommendations on how to improve cluster performance and reduce operating costs from the Amazon Redshift Advisor. For more information, see <a href="#">Amazon Redshift Advisor</a> .                                                                  | July 26, 2018      |
| <a href="#">Immediate alias reference</a>                                | You can now refer to an aliased expression immediately after you define it. For more information, see <a href="#">SELECT List</a> .                                                                                                                                              | July 18, 2018      |
| <a href="#">Specify compression type when creating an external table</a> | You can now specify compression type when creating an external table with Amazon Redshift Spectrum. For more information, see <a href="#">Create External Tables</a> .                                                                                                           | June 27, 2018      |
| <a href="#">PG_LAST_UNLOAD_ID</a>                                        | Documentation added for a new System Information function: PG_LAST_UNLOAD_ID. For more information, see <a href="#">PG_LAST_UNLOAD_ID</a> .                                                                                                                                      | June 27, 2018      |
| <a href="#">ALTER TABLE RENAME COLUMN</a>                                | ALTER TABLE now supports renaming columns for external tables. For more information, see <a href="#">Alter External Table Examples</a> .                                                                                                                                         | June 7, 2018       |

## Earlier updates

The following table describes the important changes in each release of the *Amazon Redshift Database Developer Guide* before June 2018.

| Change                                                  | Description                                                                                                                                                                                                                                                                                                                                                                                             | Date changed    |
|---------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| COPY from Parquet includes SMALLINT                     | COPY now supports loading from Parquet formatted files into columns that use the SMALLINT data type. For more information, see <a href="#">COPY from columnar data formats (p. 722)</a>                                                                                                                                                                                                                 | January 2, 2019 |
| COPY from columnar formats                              | COPY now supports loading from files on Amazon S3 that use Parquet and ORC columnar data formats. For more information, see <a href="#">COPY from columnar data formats (p. 722)</a>                                                                                                                                                                                                                    | May 17, 2018    |
| Dynamic maximum run time for SQA                        | By default, workload management (WLM) now dynamically assigns a value for the short query acceleration (SQA) maximum run time based on analysis of your cluster's workload. For more information, see <a href="#">Maximum runtime for short queries (p. 519)</a> .                                                                                                                                      | May 17, 2018    |
| New column in STL_LOAD_COMMITS                          | The <a href="#">STL_LOAD_COMMITS (p. 1523)</a> system table has a new column, <code>file_format</code> .                                                                                                                                                                                                                                                                                                | May 10, 2018    |
| New columns in STL_HASHJOIN and other system log tables | The <a href="#">STL_HASHJOIN (p. 1519)</a> system table has three new columns, <code>hash_segment</code> , <code>hash_step</code> , and <code>checksum</code> . Also, a <code>checksum</code> was added to <code>STL_MERGEJOIN</code> , <code>STL_NESTLOOP</code> , <code>STL_HASH</code> , <code>STL_SCAN</code> , <code>STL_SORT</code> , <code>STL_LIMIT</code> , and <code>STL_PROJECT</code> .     | May 17, 2018    |
| New columns in STL_AGGR                                 | The <a href="#">STL_AGGR (p. 1496)</a> system table has two new columns, <code>resizes</code> and <code>flushable</code> .                                                                                                                                                                                                                                                                              | April 19, 2018  |
| New options for REGEX functions                         | For the <a href="#">REGEXP_INSTR (p. 1302)</a> and <a href="#">REGEXP_SUBSTR (p. 1306)</a> functions, you can now specify which occurrence of a match to use and whether to perform a case-sensitive match. <code>REGEXP_INSTR</code> also allows you specify whether to return the position of the first character of the match or the position of the first character following the end of the match. | March 22, 2018  |
| New columns in system tables                            | The <code>tombstonedblocks</code> , <code>tossedblocks</code> , and <code>batched_by</code> columns were added to the <a href="#">STL_COMMIT_STATS (p. 1503)</a> system table. The <code>localslice</code> column was added to the <a href="#">STV_SLICES (p. 1598)</a> system view.                                                                                                                    | March 22, 2018  |
| Add and drop columns in external tables                 | <a href="#">ALTER TABLE (p. 640)</a> now supports ADD COLUMN and DROP COLUMN for Amazon Redshift Spectrum external tables.                                                                                                                                                                                                                                                                              | March 22, 2018  |
| Redshift Spectrum new AWS Regions                       | Redshift Spectrum is now available in the Mumbai and São Paulo Regions. For a list of supported Regions, see <a href="#">Amazon Redshift Spectrum Regions (p. 249)</a> .                                                                                                                                                                                                                                | March 22, 2018  |

| <b>Change</b>                                                         | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                         | <b>Date changed</b> |
|-----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| Table limit increased to 20,000                                       | The maximum number of tables is now 20,000 for 8xlarge cluster node types. The limit for large and xlarge node types is 9,900. For more information, see <a href="#">Limits and quotas (p. 831)</a> .                                                                                                                                                                                                                                      | March 13, 2018      |
| Redshift Spectrum support for JSON and Ion                            | Using Redshift Spectrum, you can reference files with scalar data in JSON or Ion data formats. For more information, see <a href="#">CREATE EXTERNAL TABLE (p. 768)</a> .                                                                                                                                                                                                                                                                  | February 26, 2018   |
| IAM role chaining for Redshift Spectrum                               | You can chain AWS Identity and Access Management (IAM) roles so that your cluster can assume other roles not attached to the cluster, including roles belonging to another AWS account. For more information, see <a href="#">Chaining IAM roles in Amazon Redshift Spectrum (p. 263)</a> .                                                                                                                                                | February 1, 2018    |
| ADD PARTITION supports IF NOT EXISTS                                  | The ADD PARTITION clause for ALTER TABLE now supports an IF NOT EXISTS option. For more information, see <a href="#">ALTER TABLE (p. 640)</a> .                                                                                                                                                                                                                                                                                            | January 11, 2018    |
| DATE data for external tables                                         | Redshift Spectrum external tables now support the DATE data type. For more information, see <a href="#">CREATE EXTERNAL TABLE (p. 768)</a> .                                                                                                                                                                                                                                                                                               | January 11, 2018    |
| Redshift Spectrum new AWS Regions                                     | Redshift Spectrum is now available in the Singapore, Sydney, Seoul, and Frankfurt Regions. For a list of supported AWS Regions, see <a href="#">Amazon Redshift Spectrum Regions (p. 249)</a> .                                                                                                                                                                                                                                            | November 16, 2017   |
| Short query acceleration in Amazon Redshift workload management (WLM) | Short query acceleration (SQA) prioritizes selected short-running queries ahead of longer-running queries. SQA executes short-running queries in a dedicated space, so that SQA queries aren't forced to wait in queues behind longer queries. With SQA, short-running queries begin executing more quickly and users see results sooner. For more information, see <a href="#">Working with short query acceleration (p. 518)</a> .       | November 16, 2017   |
| WLM reassigns hopped queries                                          | Instead of canceling and restarting a hopped query, Amazon Redshift workload management (WLM) now reassigns eligible queries to a new queue. When WLM reassigns a query, it moves the query to the new queue and continues execution, which saves time and system resources. Hopped queries that are not eligible to be reassigned are restarted or canceled. For more information, see <a href="#">WLM query queue hopping (p. 502)</a> . | November 16, 2017   |
| System log access for users                                           | In most system log tables that are visible to users, rows generated by another user are invisible to a regular user by default. To permit a regular user to see all rows in user-visible tables, including rows generated by another user, run <a href="#">ALTER USER (p. 659)</a> or <a href="#">CREATE USER (p. 846)</a> and set the <a href="#">SYSLOG ACCESS (p. 660)</a> parameter to UNRESTRICTED.                                   | November 16, 2017   |

| Change                                                     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Date changed      |
|------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| Result caching                                             | With <a href="#">Result caching (p. 6)</a> , when you run a query Amazon Redshift caches the result. When you run the query again, Amazon Redshift checks for a valid, cached copy of the query result. If a match is found in the result cache, Amazon Redshift uses the cached result and doesn't run the query. Result caching is turned on by default. To turn off result caching, set the <a href="#">enable_result_cache_for_session (p. 1691)</a> configuration parameter to off. | November 16, 2017 |
| Column metadata functions                                  | <a href="#">PG_GET_COLS (p. 1395)</a> and <a href="#">PG_GET_LATE_BINDING_VIEW_COLS (p. 1398)</a> return column metadata for Amazon Redshift tables, views, and late-binding views.                                                                                                                                                                                                                                                                                                      | November 16, 2017 |
| WLM queue hopping for CTAS                                 | Amazon Redshift workload management (WLM) now supports query queue hopping for <a href="#">CREATE TABLE AS (p. 838)</a> (CTAS) statements as well as read-only queries, such as SELECT statements. For more information, see <a href="#">WLM query queue hopping (p. 502)</a> .                                                                                                                                                                                                          | October 19, 2017  |
| Amazon Redshift Spectrum manifest files                    | When you create a Redshift Spectrum external table, you can specify a manifest file that lists the locations of data files on Amazon S3. For more information, see <a href="#">CREATE EXTERNAL TABLE (p. 768)</a> .                                                                                                                                                                                                                                                                      | October 19, 2017  |
| Amazon Redshift Spectrum new AWS Regions                   | Redshift Spectrum is now available in the EU (Ireland) and Asia Pacific (Tokyo) Regions. For a list of supported AWS Regions, see <a href="#">Amazon Redshift Spectrum considerations (p. 249)</a> .                                                                                                                                                                                                                                                                                     | October 19, 2017  |
| Amazon Redshift Spectrum added file formats                | You can now create Redshift Spectrum external tables based on Regex, OpenCSV, and Avro data file formats. For more information, see <a href="#">CREATE EXTERNAL TABLE (p. 768)</a> .                                                                                                                                                                                                                                                                                                     | October 5, 2017   |
| Pseudocolumns for Amazon Redshift Spectrum external tables | You can select the \$path and \$size pseudocolumns in a Redshift Spectrum external table to view the location and size of the referenced data files in Amazon S3. For more information, see <a href="#">Pseudocolumns (p. 278)</a> .                                                                                                                                                                                                                                                     | October 5, 2017   |
| Functions to validate JSON                                 | You can use the <a href="#">IS_VALID_JSON (p. 1143)</a> and <a href="#">IS_VALID_JSON_ARRAY (p. 1144)</a> functions to check for valid JSON formatting. The other JSON functions now have an optional null_if_invalid argument.                                                                                                                                                                                                                                                          | October 5, 2017   |
| LISTAGG DISTINCT                                           | You can use the DISTINCT clause with the <a href="#">LISTAGG (p. 1029)</a> aggregate function and the <a href="#">LISTAGG (p. 1354)</a> window function to eliminate duplicate values from the specified expression before concatenating.                                                                                                                                                                                                                                                | October 5, 2017   |
| View column names in uppercase                             | To view column names in SELECT results in uppercase, you can set the <a href="#">describe_field_name_in_uppercase (p. 1687)</a> configuration parameter to true.                                                                                                                                                                                                                                                                                                                         | October 5, 2017   |

| Change                                          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Date changed       |
|-------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| Skip header lines in external tables            | You can set the <code>skip.header.line.count</code> property in the <a href="#">CREATE EXTERNAL TABLE (p. 768)</a> command to skip header lines at the beginning of Redshift Spectrum data files.                                                                                                                                                                                                                                                                                                                                                                                                                                             | October 5, 2017    |
| Scan row count                                  | WLM query monitor rules uses the <code>scan_row_count</code> metric to return the number of rows in a scan step. The row count is the total number of rows emitted before filtering rows marked for deletion (ghost rows) and before applying user-defined query filters. For more information, see <a href="#">Query monitoring metrics for Amazon Redshift (p. 528)</a> .                                                                                                                                                                                                                                                                   | September 21, 2017 |
| SQL user-defined functions                      | A scalar SQL user-defined function (UDF) incorporates a SQL SELECT clause that executes when the function is called and returns a single value. For more information, see <a href="#">Creating a scalar SQL UDF (p. 151)</a> .                                                                                                                                                                                                                                                                                                                                                                                                                | August 31, 2017    |
| Late-binding views                              | A late-binding view is not bound to the underlying database objects, such as tables and user-defined functions. As a result, there is no dependency between the view and the objects it references. You can create a view even if the referenced objects don't exist. Because there is no dependency, you can drop or alter a referenced object without affecting the view. Amazon Redshift doesn't check for dependencies until the view is queried. To create a late-binding view, specify the <code>WITH NO SCHEMA BINDING</code> clause with your CREATE VIEW statement. For more information, see <a href="#">CREATE VIEW (p. 851)</a> . | August 31, 2017    |
| OCTET_LENGTH function                           | <a href="#">OCTET_LENGTH (p. 1297)</a> returns the length of the specified string as the number of bytes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | August 18, 2017    |
| ORC and Grok files types supported              | Amazon Redshift Spectrum now supports the ORC and Grok data formats for Redshift Spectrum data files. For more information, see <a href="#">Creating data files for queries in Amazon Redshift Spectrum (p. 270)</a> .                                                                                                                                                                                                                                                                                                                                                                                                                        | August 18, 2017    |
| RegexSerDe now supported                        | Amazon Redshift Spectrum now supports the RegexSerDe data format. For more information, see <a href="#">Creating data files for queries in Amazon Redshift Spectrum (p. 270)</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                            | July 19, 2017      |
| New columns added to SVV_TABLES and SVV_COLUMNS | The columns <code>domain_name</code> and <code>remarks</code> were added to <a href="#">SVV_COLUMNS (p. 1418)</a> . A <code>remarks</code> column was added to <a href="#">SVV_TABLES (p. 1456)</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                         | July 19, 2017      |
| SVV_TABLES and SVV_COLUMNS system views         | The <a href="#">SVV_TABLES (p. 1456)</a> and <a href="#">SVV_COLUMNS (p. 1418)</a> system views provide information about columns and other details for local and external tables and views.                                                                                                                                                                                                                                                                                                                                                                                                                                                  | July 7, 2017       |

| <b>Change</b>                                                                      | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                     | <b>Date changed</b> |
|------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| VPC no longer required for Amazon Redshift Spectrum with Amazon EMR Hive metastore | Redshift Spectrum removed the requirement that the Amazon Redshift cluster and the Amazon EMR cluster must be in the same VPC and the same subnet when using an Amazon EMR Hive metastore. For more information, see <a href="#">Working with external catalogs in Amazon Redshift Spectrum (p. 274)</a> .                                                                                                             | July 7, 2017        |
| UNLOAD to smaller file sizes                                                       | By default, UNLOAD creates multiple files on Amazon S3 with a maximum size of 6.2 GB. To create smaller files, specify the MAXFILESIZE with the UNLOAD command. You can specify a maximum file size between 5 MB and 6.2 GB. For more information, see <a href="#">UNLOAD (p. 991)</a> .                                                                                                                               | July 7, 2017        |
| TABLE PROPERTIES                                                                   | You can now set the TABLE PROPERTIES numRows parameter for <a href="#">CREATE EXTERNAL TABLE (p. 768)</a> or <a href="#">ALTER TABLE (p. 640)</a> to update table statistics to reflect the number of rows in the table.                                                                                                                                                                                               | June 6, 2017        |
| ANALYZE PREDICATE COLUMNS                                                          | To save time and cluster resources, you can choose to analyze only the columns that are likely to be used as predicates. When you run ANALYZE with the PREDICATE COLUMNS clause, the analyze operation includes only columns that have been used in a join, filter condition, or group by clause, or are used as a sort key or distribution key. For more information, see <a href="#">Analyzing tables (p. 105)</a> . | May 25, 2017        |
| IAM policies for Amazon Redshift Spectrum                                          | To grant access to an Amazon S3 bucket only using Redshift Spectrum, you can include a condition that allows access for the user agent AWS Redshift/Spectrum. For more information, see <a href="#">IAM policies for Amazon Redshift Spectrum (p. 259)</a> .                                                                                                                                                           | May 25, 2017        |
| Amazon Redshift Spectrum Recursive Scan                                            | Redshift Spectrum now scans files in subfolders as well as the specified folder in Amazon S3. For more information, see <a href="#">Creating external tables for Redshift Spectrum (p. 277)</a> .                                                                                                                                                                                                                      | May 25, 2017        |
| Query monitoring rules                                                             | Using WLM query monitoring rules, you can define metrics-based performance boundaries for WLM queues and specify what action to take when a query goes beyond those boundaries—log, hop, or abort. You define query monitoring rules as part of your workload management (WLM) configuration. For more information, see <a href="#">WLM query monitoring rules (p. 527)</a> .                                          | April 21, 2017      |

| <b>Change</b>                                  | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | <b>Date changed</b> |
|------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| Amazon Redshift Spectrum                       | <p>Using Redshift Spectrum, you can efficiently query and retrieve data from files in Amazon S3 without having to load the data into tables. Redshift Spectrum queries execute very fast against large datasets because Redshift Spectrum scans the data files directly in Amazon S3. Much of the processing occurs in the Amazon Redshift Spectrum layer, and most of the data remains in Amazon S3. Multiple clusters can concurrently query the same dataset on Amazon S3 without the need to make copies of the data for each cluster. For more information, see <a href="#">Querying external data using Amazon Redshift Spectrum (p. 248)</a></p> | April 19, 2017      |
| New system tables to support Redshift Spectrum | <p>The following new system views have been added to support Redshift Spectrum:</p> <ul style="list-style-type: none"> <li>• <a href="#">SVL_S3QUERY (p. 1653)</a></li> <li>• <a href="#">SVL_S3QUERY_SUMMARY (p. 1655)</a></li> <li>• <a href="#">SVV_EXTERNAL_COLUMNS (p. 1428)</a></li> <li>• <a href="#">SVV_EXTERNAL_DATABASES (p. 1429)</a></li> <li>• <a href="#">SVV_EXTERNAL_PARTITIONS (p. 1429)</a></li> <li>• <a href="#">SVV_EXTERNAL_TABLES (p. 1431)</a></li> <li>• <a href="#">PG_EXTERNAL_SCHEMA (p. 1674)</a></li> </ul>                                                                                                              | April 19, 2017      |
| APPROXIMATE PERCENTILE_DISC aggregate function | <p>The <a href="#">APPROXIMATE PERCENTILE_DISC (p. 1024)</a> aggregate function is now available.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | April 4, 2017       |
| Server-side encryption with KMS                | <p>You can now unload data to Amazon S3 using server-side encryption with an AWS Key Management Service key (SSE-KMS). In addition, <a href="#">COPY (p. 677)</a> now transparently loads KMS-encrypted data files from Amazon S3. For more information, see <a href="#">UNLOAD (p. 991)</a>.</p>                                                                                                                                                                                                                                                                                                                                                       | February 9, 2017    |
| New authorization syntax                       | <p>You can now use the IAM_ROLE, MASTER_SYMMETRIC_KEY, ACCESS_KEY_ID, SECRET_ACCESS_KEY, and SESSION_TOKEN parameters to provide authorization and access information for COPY, UNLOAD, and CREATE LIBRARY commands. The new authorization syntax provides a more flexible alternative to providing a single string argument to the CREDENTIALS parameter. For more information, see <a href="#">Authorization parameters (p. 693)</a>.</p>                                                                                                                                                                                                             | February 9, 2017    |
| Schema limit increase                          | <p>You can now create up to 9,900 schemas per cluster. For more information, see <a href="#">CREATE SCHEMA (p. 821)</a>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | February 9, 2017    |
| Default table encoding                         | <p><a href="#">CREATE TABLE (p. 823)</a> and <a href="#">ALTER TABLE (p. 640)</a> now assign LZO compression encoding to most new columns. Columns defined as sort keys, columns that are defined as BOOLEAN, REAL, or DOUBLE PRECISION data types, and temporary tables are assigned RAW encoding by default. For more information, see <a href="#">ENCODE (p. 826)</a>.</p>                                                                                                                                                                                                                                                                           | February 6, 2017    |

| <b>Change</b>                                     | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                        | <b>Date changed</b> |
|---------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| ZSTD compression encoding                         | Amazon Redshift now supports <a href="#">ZSTD (p. 45)</a> column compression encoding.                                                                                                                                                                                                                                                                                                                    | January 19, 2017    |
| PERCENTILE_CONT and MEDIAN aggregate functions    | <a href="#">PERCENTILE_CONT (p. 1035)</a> and <a href="#">MEDIAN (p. 1032)</a> are now available as aggregate functions as well as window functions.                                                                                                                                                                                                                                                      | January 19, 2017    |
| User-defined function (UDF) User Logging          | You can use the Python logging module to create user-defined error and warning messages in your UDFs. Following query execution, you can query the <a href="#">SVL_UDF_LOG (p. 1667)</a> system view to retrieve logged messages. For more information about user-defined messages, see <a href="#">Logging errors and warnings in UDFs (p. 162)</a> .                                                    | December 8, 2016    |
| ANALYZE COMPRESSION estimated reduction           | The ANALYZE COMPRESSION command now reports an estimate for percentage reduction in disk space for each column. For more information, see <a href="#">ANALYZE COMPRESSION (p. 665)</a> .                                                                                                                                                                                                                  | November 10, 2016   |
| Connection limits                                 | You can now set a limit on the number of database connections a user is permitted to have open concurrently. You can also limit the number of concurrent connections for a database. For more information, see <a href="#">CREATE USER (p. 846)</a> and <a href="#">CREATE DATABASE (p. 745)</a> .                                                                                                        | November 10, 2016   |
| COPY sort order enhancement                       | COPY now automatically adds new rows to the table's sorted region when you load your data in sort key order. For specific requirements to enable this enhancement, see <a href="#">Loading your data in sort key order (p. 117)</a> .                                                                                                                                                                     | November 10, 2016   |
| CTAS with compression                             | CREATE TABLE AS (CTAS) now automatically assigns compression encodings to new tables based on the column's data type. For more information, see <a href="#">Inheritance of column and table attributes (p. 841)</a> .                                                                                                                                                                                     | October 28, 2016    |
| Time stamp with time zone data type               | Amazon Redshift now supports a timestamp with time zone ( <a href="#">TIMESTAMPTZ (p. 586)</a> ) data type. Also, several new functions have been added to support the new data type. For more information, see <a href="#">Date and time functions (p. 1086)</a> .                                                                                                                                       | September 29, 2016  |
| Analyze threshold                                 | To reduce processing time and improve overall system performance for <a href="#">ANALYZE (p. 663)</a> operations, Amazon Redshift skips analyzing a table if the percentage of rows that have changed since the last ANALYZE command run is lower than the analyze threshold specified by the <a href="#">analyze_threshold_percent (p. 1685)</a> parameter. By default, analyze_threshold_percent is 10. | August 9, 2016      |
| New STL_RESTARTED_SESSIONS system table           | When Amazon Redshift restarts a session, <a href="#">STL_RESTARTED_SESSIONS (p. 1547)</a> records the new process ID (PID) and the old PID.                                                                                                                                                                                                                                                               | August 9, 2016      |
| Updated the Date and Time Functions documentation | Added a summary of functions with links to the <a href="#">Date and time functions (p. 1086)</a> , and updated the function references for consistency.                                                                                                                                                                                                                                                   | June 24, 2016       |

| Change                                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Date changed   |
|------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| New columns in <code>STL_CONNECTION_LOG</code> | The <a href="#">STL_CONNECTION_LOG (p. 1505)</a> system table has two new columns to track SSL connections. If you routinely load audit logs to an Amazon Redshift table, you will need to add the following new columns to the target table: <code>sslcompression</code> and <code>sslexpansion</code> .                                                                                                                                                                                                                                        | May 5, 2016    |
| MD5-hash password                              | You can specify the password for a <a href="#">CREATE USER (p. 846)</a> or <a href="#">ALTER USER (p. 659)</a> command by supplying the MD5-hash string of the password and user name.                                                                                                                                                                                                                                                                                                                                                           | April 21, 2016 |
| New column in <code>STV_TBL_PERM</code>        | The backup column in the <a href="#">STV_TBL_PERM (p. 1599)</a> system view indicates whether the table is included in cluster snapshots. For more information, see <a href="#">BACKUP (p. 829)</a> .                                                                                                                                                                                                                                                                                                                                            | April 21, 2016 |
| No-backup tables                               | For tables, such as staging tables, that won't contain critical data, you can specify <code>BACKUP NO</code> in your <a href="#">CREATE TABLE (p. 823)</a> or <a href="#">CREATE TABLE AS (p. 838)</a> statement to prevent Amazon Redshift from including the table in automated or manual snapshots. Using no-backup tables saves processing time when creating snapshots and restoring from snapshots and reduces storage space on Amazon S3.                                                                                                 | April 7, 2016  |
| VACUUM delete threshold                        | By default, the <a href="#">VACUUM (p. 1015)</a> command now reclaims space such that at least 95 percent of the remaining rows are not marked for deletion. As a result, VACUUM usually needs much less time for the delete phase compared to reclaiming 100 percent of deleted rows. You can change the default threshold for a single table by including the <code>TO threshold PERCENT</code> parameter when you run the VACUUM command.                                                                                                     | April 7, 2016  |
| <code>SVV_TRANSACTIONS</code> system table     | The <a href="#">SVV_TRANSACTIONS (p. 1460)</a> system view records information about transactions that currently hold locks on tables in the database.                                                                                                                                                                                                                                                                                                                                                                                           | April 7, 2016  |
| Using IAM roles to access other AWS resources  | To move data between your cluster and another AWS resource, such as Amazon S3, DynamoDB, Amazon EMR, or Amazon EC2, your cluster must have permission to access the resource and perform the necessary actions. As a more secure alternative to providing an access key pair with <code>COPY</code> , <code>UNLOAD</code> , or <code>CREATE LIBRARY</code> commands, you can now specify an IAM role that your cluster uses for authentication and authorization. For more information, see <a href="#">Role-based access control (p. 715)</a> . | March 29, 2016 |
| VACUUM sort threshold                          | The VACUUM command now skips the sort phase for any table where more than 95 percent of the table's rows are already sorted. You can change the default sort threshold for a single table by including the <code>TO threshold PERCENT</code> parameter when you run the <a href="#">VACUUM (p. 1015)</a> command.                                                                                                                                                                                                                                | March 17, 2016 |

| <b>Change</b>                                  | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                              | <b>Date changed</b> |
|------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| New columns in <code>STL_CONNECTION_LOG</code> | The <a href="#">STL_CONNECTION_LOG (p. 1505)</a> system table has three new columns. If you routinely load audit logs to an Amazon Redshift table, you will need to add the following new columns to the target table: <code>sslversion</code> , <code>sslcipher</code> , and <code>mtu</code> .                                                                                                | March 17, 2016      |
| UNLOAD with bzip2 compression                  | You now have the option to <a href="#">UNLOAD (p. 991)</a> using bzip2 compression.                                                                                                                                                                                                                                                                                                             | February 8, 2016    |
| ALTER TABLE APPEND                             | <a href="#">ALTER TABLE APPEND (p. 656)</a> appends rows to a target table by moving data from an existing source table. <code>ALTER TABLE APPEND</code> is usually much faster than a similar <a href="#">CREATE TABLE AS (p. 838)</a> or <a href="#">INSERT (p. 902)</a> <code>INTO</code> operation because data is moved, not duplicated.                                                   | February 8, 2016    |
| WLM query queue hopping                        | If workload management (WLM) cancels a read-only query, such as a <code>SELECT</code> statement, due to a WLM timeout, WLM attempts to route the query to the next matching queue. For more information, see <a href="#">WLM query queue hopping (p. 502)</a>                                                                                                                                   | January 7, 2016     |
| ALTER DEFAULT PRIVILEGES                       | You can use the <a href="#">ALTER DEFAULT PRIVILEGES (p. 631)</a> command to define the default set of access privileges to be applied to objects that are created in the future by the specified user.                                                                                                                                                                                         | December 10, 2015   |
| bzip2 file compression                         | The <a href="#">COPY (p. 677)</a> command supports loading data from files that were compressed using bzip2.                                                                                                                                                                                                                                                                                    | December 10, 2015   |
| NULLS FIRST and NULLS LAST                     | You can specify whether an <code>ORDER BY</code> clause should rank <code>NULLS</code> first or last in the result set. For more information, see <a href="#">ORDER BY clause (p. 969)</a> and <a href="#">Window function syntax summary (p. 1336)</a> .                                                                                                                                       | November 19, 2015   |
| REGION keyword for CREATE LIBRARY              | If the Amazon S3 bucket that contains the UDF library files does not reside in the same AWS Region as your Amazon Redshift cluster, you can use the <code>REGION</code> option to specify the region in which the data is located. For more information, see <a href="#">CREATE LIBRARY (p. 793)</a> .                                                                                          | November 19, 2015   |
| User-defined scalar functions (UDFs)           | You can now create custom user-defined scalar functions to implement non-SQL processing functionality provided either by Amazon Redshift-supported modules in the Python 2.7 Standard Library or your own custom UDFs based on the Python programming language. For more information, see <a href="#">Creating user-defined functions (p. 150)</a> .                                            | September 11, 2015  |
| Dynamic properties in WLM configuration        | The WLM configuration parameter now supports applying some properties dynamically. Other properties remain static changes and require that associated clusters be rebooted so that the configuration changes can be applied. For more information, see <a href="#">WLM dynamic and static configuration properties (p. 524)</a> and <a href="#">Implementing workload management (p. 491)</a> . | August 3, 2015      |

| Change                                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Date changed      |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| LISTAGG function                        | The <a href="#">LISTAGG function (p. 1029)</a> and <a href="#">LISTAGG window function (p. 1354)</a> return a string created by concatenating a set of column values.                                                                                                                                                                                                                                                                                                                                                           | July 30, 2015     |
| Deprecated parameter                    | The <i>max_cursor_result_set_size</i> configuration parameter is deprecated. The size of cursor result sets are constrained based on the cluster's node type. For more information, see <a href="#">Cursor constraints (p. 856)</a> .                                                                                                                                                                                                                                                                                           | July 24, 2015     |
| Revised COPY command documentation      | The <a href="#">COPY (p. 677)</a> command reference has been extensively revised to make the material friendlier and more accessible.                                                                                                                                                                                                                                                                                                                                                                                           | July 15, 2015     |
| COPY from Avro format                   | The <a href="#">COPY (p. 677)</a> command supports loading data in Avro format from data files on Amazon S3, Amazon EMR, and from remote hosts using SSH. For more information, see <a href="#">AVRO (p. 698)</a> and <a href="#">Copy from Avro examples (p. 738)</a> .                                                                                                                                                                                                                                                        | July 8, 2015      |
| STV_STARTUP_RECOVERY_STATE              | <del>RETRY STATE</del> <a href="#">STARTUP_RECOVERY_STATE (p. 1599)</a> system table records the state of tables that are temporarily locked during cluster restart operations. Amazon Redshift places a temporary lock on tables while they are being processed to resolve stale transactions following a cluster restart.                                                                                                                                                                                                     | May 25, 2015      |
| ORDER BY optional for ranking functions | The ORDER BY clause is now optional for certain window ranking functions. For more information, see <a href="#">CUME_DIST window function (p. 1344)</a> , <a href="#">DENSE_RANK window function (p. 1345)</a> , <a href="#">RANK window function (p. 1369)</a> , <a href="#">NTILE window function (p. 1363)</a> , <a href="#">PERCENT_RANK window function (p. 1364)</a> , and <a href="#">ROW_NUMBER window function (p. 1373)</a> .                                                                                         | May 25, 2015      |
| Interleaved sort keys                   | Interleaved sort keys give equal weight to each column in the sort key. Using interleaved sort keys instead of the default compound keys significantly improves performance for queries that use restrictive predicates on secondary sort columns, especially for large tables. Interleaved sorting also improves overall performance when multiple queries filter on different columns in the same table. For more information, see <a href="#">Working with sort keys (p. 60)</a> and <a href="#">CREATE TABLE (p. 823)</a> . | May 11, 2015      |
| Revised tuning query performance topic  | <a href="#">Tuning query performance (p. 463)</a> has been expanded to include new queries for analyzing query performance and more examples. Also, the topic has been revised to be clearer and more complete. <a href="#">Amazon Redshift best practices for designing queries (p. 21)</a> has more information about how to write queries to improve performance.                                                                                                                                                            | March 23, 2015    |
| SVL_QUERY_QUEUE_INFO                    | <del>INFO</del> <a href="#">SVL_QUERY_QUEUE_INFO (p. 1644)</a> view summarizes details for queries that spent time in a WLM query queue or commit queue.                                                                                                                                                                                                                                                                                                                                                                        | February 19, 2015 |

| Change                                                                          | Description                                                                                                                                                                                                                                                                                                                                                           | Date changed      |
|---------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| SVV_TABLE_INFO                                                                  | You can use the <a href="#">SVV_TABLE_INFO (p. 1457)</a> view to diagnose and address table design issues that can influence query performance, including issues with compression encoding, distribution keys, sort style, data distribution skew, table size, and statistics.                                                                                        | February 19, 2015 |
| UNLOAD uses server-side file encryption                                         | The <a href="#">UNLOAD (p. 991)</a> command now automatically uses Amazon S3 server-side encryption (SSE) to encrypt all unload data files. Server-side encryption adds another layer of data security with little or no change in performance.                                                                                                                       | October 31, 2014  |
| CUME_DIST window function                                                       | The <a href="#">CUME_DIST window function (p. 1344)</a> calculates the cumulative distribution of a value within a window or partition.                                                                                                                                                                                                                               | October 31, 2014  |
| MONTHS_BETWEEN function                                                         | The <a href="#">MONTHS_BETWEEN function (p. 1117)</a> determines the number of months between two dates.                                                                                                                                                                                                                                                              | October 31, 2014  |
| NEXT_DAY function                                                               | The <a href="#">NEXT_DAY function (p. 1118)</a> returns the date of the first instance of a specified day that is later than the given date.                                                                                                                                                                                                                          | October 31, 2014  |
| PERCENT_RANK window function                                                    | The <a href="#">PERCENT_RANK window function (p. 1364)</a> calculates the percent rank of a given row.                                                                                                                                                                                                                                                                | October 31, 2014  |
| RATIO_TO_REPORT window function                                                 | The <a href="#">RATIO_TO_REPORT window function (p. 1371)</a> calculates the ratio of a value to the sum of the values in a window or partition.                                                                                                                                                                                                                      | October 31, 2014  |
| TRANSLATE function                                                              | The <a href="#">TRANSLATE function (p. 1321)</a> replaces all occurrences of specified characters within a given expression with specified substitutes.                                                                                                                                                                                                               | October 31, 2014  |
| NVL2 function                                                                   | The <a href="#">NVL2 function (p. 1063)</a> returns one of two values based on whether a specified expression evaluates to NULL or NOT NULL.                                                                                                                                                                                                                          | October 16, 2014  |
| MEDIAN window function                                                          | The <a href="#">MEDIAN window function (p. 1358)</a> calculates the median value for the range of values in a window or partition.                                                                                                                                                                                                                                    | October 16, 2014  |
| ON ALL TABLES IN SCHEMA <i>schema_name</i> clause for GRANT and REVOKE commands | The <a href="#">GRANT (p. 887)</a> and <a href="#">REVOKE (p. 916)</a> commands have been updated with an ON ALL TABLES IN SCHEMA <i>schema_name</i> clause. This clause allows you to use a single command to change privileges for all tables in a schema.                                                                                                          | October 16, 2014  |
| IF EXISTS clause for DROP SCHEMA, DROP TABLE, DROP USER, and DROP VIEW commands | The <a href="#">DROP SCHEMA (p. 872)</a> , <a href="#">DROP TABLE (p. 874)</a> , <a href="#">DROP USER (p. 876)</a> , and <a href="#">DROP VIEW (p. 878)</a> commands have been updated with an IF EXISTS clause. This clause causes the command to make no changes and return a message rather than terminating with an error if the specified object doesn't exist. | October 16, 2014  |

| Change                                                           | Description                                                                                                                                                                                                                                                                                                                 | Date changed       |
|------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| IF NOT EXISTS clause for CREATE SCHEMA and CREATE TABLE commands | The <a href="#">CREATE SCHEMA (p. 821)</a> and <a href="#">CREATE TABLE (p. 823)</a> commands have been updated with an IF NOT EXISTS clause. This clause causes the command to make no changes and return a message rather than terminating with an error if the specified object already exists.                          | October 16, 2014   |
| COPY support for UTF-16 encoding                                 | The COPY command now supports loading from data files that use UTF-16 encoding as well as UTF-8 encoding. For more information, see <a href="#">ENCODING (p. 707)</a> .                                                                                                                                                     | September 29, 2014 |
| New workload management tutorial                                 | <a href="#">Tutorial: Configuring manual workload management (WLM) queues (p. 505)</a> walks you through the process of configuring Workload Management (WLM) queues to improve query processing and resource allocation.                                                                                                   | September 25, 2014 |
| AES 128-bit encryption                                           | The COPY command now supports AES 128-bit encryption as well as AES 256-bit encryption when loading from data files encrypted using Amazon S3 client-side encryption. For more information, see <a href="#">Loading encrypted data files from Amazon S3 (p. 73)</a> .                                                       | September 29, 2014 |
| PG_LAST_UNLOAD_COUNT function                                    | The PG_LAST_UNLOAD_COUNT function returns the number of rows that were processed in the most recent UNLOAD operation. For more information, see <a href="#">PG_LAST_UNLOAD_COUNT (p. 1402)</a> .                                                                                                                            | September 15, 2014 |
| New troubleshooting queries section                              | <a href="#">Troubleshooting queries (p. 486)</a> provides a quick reference for identifying and addressing some of the most common and most serious issues you are likely to encounter with Amazon Redshift queries.                                                                                                        | July 7, 2014       |
| New loading data tutorial                                        | <a href="#">Tutorial: Loading data from Amazon S3 (p. 124)</a> walks you through the process of loading data into your Amazon Redshift database tables from data files in an Amazon S3 bucket, from beginning to end.                                                                                                       | July 1, 2014       |
| PERCENTILE_CONT window function                                  | <a href="#">PERCENTILE_CONT window function (p. 1365)</a> is an inverse distribution function that assumes a continuous distribution model. It takes a percentile value and a sort specification, and returns an interpolated value that would fall into the given percentile value with respect to the sort specification. | June 30, 2014      |
| PERCENTILE_DISC window function                                  | <a href="#">PERCENTILE_DISC window function (p. 1368)</a> is an inverse distribution function that assumes a discrete distribution model. It takes a percentile value and a sort specification and returns an element from the set.                                                                                         | June 30, 2014      |
| GREATEST and LEAST functions                                     | The <a href="#">GREATEST and LEAST functions (p. 1060)</a> functions return the largest or smallest value from a list of expressions.                                                                                                                                                                                       | June 30, 2014      |
| Cross-region COPY                                                | The <a href="#">COPY (p. 677)</a> command supports loading data from an Amazon S3 bucket or Amazon DynamoDB table that is located in a different region than the Amazon Redshift cluster. For more information, see <a href="#">REGION (p. 686)</a> in the COPY command reference.                                          | June 30, 2014      |

| <b>Change</b>                                     | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | <b>Date changed</b> |
|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| Best Practices expanded                           | <a href="#">Amazon Redshift best practices (p. 10)</a> has been expanded, reorganized, and moved to the top of the navigation hierarchy to make it more discoverable.                                                                                                                                                                                                                                                                                                                                                                                                      | May 28, 2014        |
| UNLOAD to a single file                           | The <a href="#">UNLOAD (p. 991)</a> command can optionally unload table data serially to a single file on Amazon S3 by adding the PARALLEL OFF option. If the size of the data is greater than the maximum file size of 6.2 GB, UNLOAD creates additional files.                                                                                                                                                                                                                                                                                                           | May 6, 2014         |
| REGEXP functions                                  | The <a href="#">REGEXP_COUNT (p. 1301)</a> , <a href="#">REGEXP_INSTR (p. 1302)</a> , and <a href="#">REGEXP_REPLACE (p. 1304)</a> functions manipulate strings based on regular expression pattern matching.                                                                                                                                                                                                                                                                                                                                                              | May 6, 2014         |
| COPY from Amazon EMR                              | The <a href="#">COPY (p. 677)</a> command supports loading data directly from Amazon EMR clusters. For more information, see <a href="#">Loading data from Amazon EMR (p. 74)</a> .                                                                                                                                                                                                                                                                                                                                                                                        | April 18, 2014      |
| WLM concurrency limit increase                    | You can now configure workload management (WLM) to run up to 50 queries concurrently in user-defined query queues. This increase gives users more flexibility for managing system performance by modifying WLM configurations. For more information, see <a href="#">Implementing manual WLM (p. 499)</a>                                                                                                                                                                                                                                                                  | April 18, 2014      |
| New configuration parameter to manage cursor size | The max_cursor_result_set_size configuration parameter defines the maximum size of data, in megabytes, that can be returned per cursor result set of a larger query. This parameter value also affects the number of concurrent cursors for the cluster, enabling you to configure a value that increases or decreases the number of cursors for your cluster.<br><br>For more information, see <a href="#">DECLARE (p. 855)</a> in this guide and <a href="#">Configure Maximum Size of a Cursor Result Set</a> in the <a href="#">Amazon Redshift Management Guide</a> . | March 28, 2014      |
| COPY from JSON format                             | The <a href="#">COPY (p. 677)</a> command supports loading data in JSON format from data files on Amazon S3 and from remote hosts using SSH. For more information, see <a href="#">COPY from JSON format (p. 719)</a> usage notes.                                                                                                                                                                                                                                                                                                                                         | March 25, 2014      |
| New system table STL_PLAN_INFO                    | The <a href="#">STL_PLAN_INFO (p. 1536)</a> table supplements the EXPLAIN command as another way to look at query plans.                                                                                                                                                                                                                                                                                                                                                                                                                                                   | March 25, 2014      |
| New function REGEXP_SUBSTR                        | The <a href="#">REGEXP_SUBSTR function (p. 1306)</a> returns the characters extracted from a string by searching for a regular expression pattern.                                                                                                                                                                                                                                                                                                                                                                                                                         | March 25, 2014      |
| New columns for STL_COMMIT_STATS                  | The <a href="#">STL_COMMIT_STATS (p. 1503)</a> table has two new columns: numxids and oldestxid.                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | March 6, 2014       |
| COPY from SSH support for gzip and lzop           | The <a href="#">COPY (p. 677)</a> command supports gzip and lzop compression when loading data through an SSH connection.                                                                                                                                                                                                                                                                                                                                                                                                                                                  | February 13, 2014   |

| Change                                    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Date changed      |
|-------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| New functions                             | The <a href="#">ROW_NUMBER window function (p. 1373)</a> returns the number of the current row. The <a href="#">STRTOINT function (p. 1317)</a> converts a string expression of a number of the specified base to the equivalent integer value. <a href="#">PG_CANCEL_BACKEND (p. 1383)</a> and <a href="#">PG_TERMINATE_BACKEND (p. 1384)</a> enable users to cancel queries and session connections. The <a href="#">LAST_DAY (p. 1116)</a> function has been added for Oracle compatibility.                   | February 13, 2014 |
| New system table                          | The <a href="#">STL_COMMIT_STATS (p. 1503)</a> system table provides metrics related to commit performance, including the timing of the various stages of commit and the number of blocks committed.                                                                                                                                                                                                                                                                                                              | February 13, 2014 |
| FETCH with single-node clusters           | When using a cursor on a single-node cluster, the maximum number of rows that can be fetched using the <a href="#">FETCH (p. 886)</a> command is 1000. <a href="#">FETCH FORWARD ALL</a> is not supported for single-node clusters.                                                                                                                                                                                                                                                                               | February 13, 2014 |
| DS_DIST_ALL_INNER redistribution strategy | DS_DIST_ALL_INNER in the Explain plan output indicates that the entire inner table was redistributed to a single slice because the outer table uses DISTSTYLE ALL. For more information, see <a href="#">Join type examples (p. 468)</a> and <a href="#">Evaluating the query plan (p. 54)</a> .                                                                                                                                                                                                                  | January 13, 2014  |
| New system tables for queries             | Amazon Redshift has added new system tables that customers can use to evaluate query execution for tuning and troubleshooting. For more information, see <a href="#">SVL_COMPILE (p. 1630)</a> , <a href="#">STL_SCAN (p. 1554)</a> , <a href="#">STL_RETURN (p. 1548)</a> , <a href="#">STL_SAVE (p. 1553)</a> , <a href="#">STL_ALERT_EVENT_LOG (p. 1498)</a> .                                                                                                                                                 | January 13, 2014  |
| Single-node cursors                       | Cursors are now supported for single-node clusters. A single-node cluster can have two cursors open at a time, with a maximum result set of 32 GB. On a single-node cluster, we recommend setting the ODBC Cache Size parameter to 1,000. For more information, see <a href="#">DECLARE (p. 855)</a> .                                                                                                                                                                                                            | December 13, 2013 |
| ALL distribution style                    | ALL distribution can dramatically shorten execution times for certain types of queries. When a table uses ALL distribution style, a copy of the table is distributed to every node. Because the table is effectively collocated with every other table, no redistribution is needed during query execution. ALL distribution is not appropriate for all tables because it increases storage requirements and load time. For more information, see <a href="#">Working with data distribution styles (p. 49)</a> . | November 11, 2013 |

| <b>Change</b>                                      | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | <b>Date changed</b> |
|----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| COPY from remote hosts                             | In addition to loading tables from data files on Amazon S3 and from Amazon DynamoDB tables, the COPY command can load text data from Amazon EMR clusters, Amazon EC2 instances, and other remote hosts by using SSH connections. Amazon Redshift uses multiple simultaneous SSH connections to read and load data in parallel. For more information, see <a href="#">Loading data from remote hosts (p. 78)</a> .                                                                      | November 11, 2013   |
| WLM memory percent used                            | You can balance workload by designating a specific percentage of memory for each queue in your workload management (WLM) configuration. For more information, see <a href="#">Implementing manual WLM (p. 499)</a> .                                                                                                                                                                                                                                                                   | November 11, 2013   |
| APPROXIMATE COUNT(DISTINCT)                        | Queries that use APPROXIMATE COUNT(DISTINCT) execute much faster, with a relative error of about 2%. The APPROXIMATE COUNT(DISTINCT) function uses a HyperLogLog algorithm. For more information, see the <a href="#">COUNT function (p. 1027)</a> .                                                                                                                                                                                                                                   | November 11, 2013   |
| New SQL functions to retrieve recent query details | Four new SQL functions retrieve details about recent queries and COPY commands. The new functions make it easier to query the system log tables, and in many cases provide necessary details without needing to access the system tables. For more information, see <a href="#">PG_BACKEND_PID (p. 1394)</a> , <a href="#">PG_LAST_COPY_ID (p. 1400)</a> , <a href="#">PG_LAST_COPY_COUNT (p. 1399)</a> , and <a href="#">PG_LAST_QUERY_ID (p. 1401)</a> .                             | November 1, 2013    |
| MANIFEST option for UNLOAD                         | The MANIFEST option for the UNLOAD command complements the MANIFEST option for the COPY command. Using the MANIFEST option with UNLOAD automatically creates a manifest file that explicitly lists the data files that were created on Amazon S3 by the unload operation. You can then use the same manifest file with a COPY command to load the data. For more information, see <a href="#">Unloading data to Amazon S3 (p. 144)</a> and <a href="#">UNLOAD examples (p. 1000)</a> . | November 1, 2013    |
| MANIFEST option for COPY                           | You can use the MANIFEST option with the <a href="#">COPY (p. 677)</a> command to explicitly list the data files that will be loaded from Amazon S3.                                                                                                                                                                                                                                                                                                                                   | October 18, 2013    |
| System tables for troubleshooting queries          | Added documentation for system tables that are used to troubleshoot queries. The <a href="#">STL_views_for_logging (p. 1494)</a> section now contains documentation for the following system tables: STL_AGGR, STL_BCAST, STL_DIST, STL_DELETE, STL_HASH, STL_HASHJOIN, STL_INSERT, STL_LIMIT, STL_MERGE, STL_MERGEJOIN, STL_NESTLOOP, STL_PARSE, STL_PROJECT, STL_SCAN, STL_SORT, STL_UNIQUE, STL_WINDOW.                                                                             | October 3, 2013     |
| CONVERT_TIMEZONE function                          | The <a href="#">CONVERT_TIMEZONE function (p. 1095)</a> converts a timestamp from one time zone to another, with the option to automatically adjust for daylight savings time.                                                                                                                                                                                                                                                                                                         | October 3, 2013     |

| <b>Change</b>                                  | <b>Description</b>                                                                                                                                                                                                                                                                                                            | <b>Date changed</b> |
|------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| SPLIT_PART function                            | The <a href="#">SPLIT_PART function (p. 1314)</a> splits a string on the specified delimiter and returns the part at the specified position.                                                                                                                                                                                  | October 3, 2013     |
| STL_USERLOG system table                       | <a href="#">STL_USERLOG (p. 1566)</a> records details for changes that occur when a database user is created, altered, or deleted.                                                                                                                                                                                            | October 3, 2013     |
| LZO column encoding and LZOP file compression. | <a href="#">LZO (p. 43)</a> column compression encoding combines a very high compression ratio with good performance. COPY from Amazon S3 supports loading from files compressed using <a href="#">LZOP (p. 706)</a> compression.                                                                                             | September 19, 2013  |
| JSON, regular expressions, and cursors         | Added support for parsing JSON strings, pattern matching using regular expressions, and using cursors to retrieve large data sets over an ODBC connection. For more information, see <a href="#">JSON functions (p. 1142)</a> , <a href="#">Pattern-matching conditions (p. 611)</a> , and <a href="#">DECLARE (p. 855)</a> . | September 10, 2013  |
| ACCEPTINVCHAR option for COPY                  | You can successfully load data that contains invalid UTF-8 characters by specifying the ACCEPTINVCHAR option with the <a href="#">COPY (p. 677)</a> command.                                                                                                                                                                  | August 29, 2013     |
| CSV option for COPY                            | The <a href="#">COPY (p. 677)</a> command now supports loading from CSV formatted input files.                                                                                                                                                                                                                                | August 9, 2013      |
| CRC32                                          | The <a href="#">CRC32 function (p. 1288)</a> performs cyclic redundancy checks.                                                                                                                                                                                                                                               | August 9, 2013      |
| WLM wildcards                                  | Workload management (WLM) supports wildcards for adding user groups and query groups to queues. For more information, see <a href="#">Wildcards (p. 501)</a> .                                                                                                                                                                | August 1, 2013      |
| WLM timeout                                    | To limit the amount of time that queries in a given WLM queue are permitted to use, you can set the WLM timeout value for each queue. For more information, see <a href="#">WLM timeout (p. 502)</a> .                                                                                                                        | August 1, 2013      |
| New COPY options 'auto' and 'epochsecs'        | The <a href="#">COPY (p. 677)</a> command performs automatic recognition of date and time formats. New time formats, 'epochsecs' and 'epochmillisecs' enable COPY to load data in epoch format.                                                                                                                               | July 25, 2013       |
| CONVERT_TIMEZONE function                      | The <a href="#">CONVERT_TIMEZONE function (p. 1095)</a> converts a timestamp from one timezone to another.                                                                                                                                                                                                                    | July 25, 2013       |
| FUNC_SHA1 function                             | The <a href="#">FUNC_SHA1 function (p. 1132)</a> converts a string using the SHA1 algorithm.                                                                                                                                                                                                                                  | July 15, 2013       |
| max_execution_time                             | To limit the amount of time queries are permitted to use, you can set the max_execution_time parameter as part of the WLM configuration. For more information, see <a href="#">Modifying the WLM configuration (p. 492)</a> .                                                                                                 | July 22, 2013       |
| Four-byte UTF-8 characters                     | The VARCHAR data type now supports four-byte UTF-8 characters. Five-byte or longer UTF-8 characters are not supported. For more information, see <a href="#">Storage and ranges (p. 582)</a> .                                                                                                                                | July 18, 2013       |

| <b>Change</b>                  | <b>Description</b>                                                                                                                                                                                                                                                             | <b>Date changed</b> |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| SVL_QERROR                     | The SVL_QERROR system view has been deprecated.                                                                                                                                                                                                                                | July 12, 2013       |
| Revised Document History       | The Document History page now shows the date the documentation was updated.                                                                                                                                                                                                    | July 12, 2013       |
| STL_UNLOAD_LOG                 | <a href="#">STL_UNLOAD_LOG (p. 1564)</a> records the details for an unload operation.                                                                                                                                                                                          | July 5, 2013        |
| JDBC fetch size parameter      | To avoid client-side out of memory errors when retrieving large data sets using JDBC, you can enable your client to fetch data in batches by setting the JDBC fetch size parameter. For more information, see <a href="#">Setting the JDBC fetch size parameter (p. 489)</a> . | June 27, 2013       |
| UNLOAD encrypted files         | <a href="#">UNLOAD (p. 991)</a> now supports unloading table data to encrypted files on Amazon S3.                                                                                                                                                                             | May 22, 2013        |
| Temporary credentials          | <a href="#">COPY (p. 677)</a> and <a href="#">UNLOAD (p. 991)</a> now support the use of temporary credentials.                                                                                                                                                                | April 11, 2013      |
| Added clarifications           | Clarified and expanded discussions of Designing Tables and Loading Data.                                                                                                                                                                                                       | February 14, 2013   |
| Added best practices           | Added <a href="#">Amazon Redshift best practices for designing tables (p. 15)</a> and <a href="#">Amazon Redshift best practices for loading data (p. 17)</a> .                                                                                                                | February 14, 2013   |
| Clarified password constraints | Clarified password constraints for CREATE USER and ALTER USER, various minor revisions.                                                                                                                                                                                        | February 14, 2013   |
| New guide                      | This is the first release of the <i>Amazon Redshift Developer Guide</i> .                                                                                                                                                                                                      | February 14, 2013   |