Amazon Redshift

Table, Query Design, and Data Distribution with Apollo Tyres team

Tejal Rathod – Solutions Architect, AWS

17th December 2021

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Discussion Topics

- Data Modelling & Queries
- Data Loads & Unloads
- Query Tuning
- Table Defragmentation/Organization
- Table Statistics

> **Bradley Todd**
> **Liberty Mutual, Technology Architect**
>
> Redshift allows us to quickly spin up clusters and provide our data scientists with a fast and easy method to access data and generate insights

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

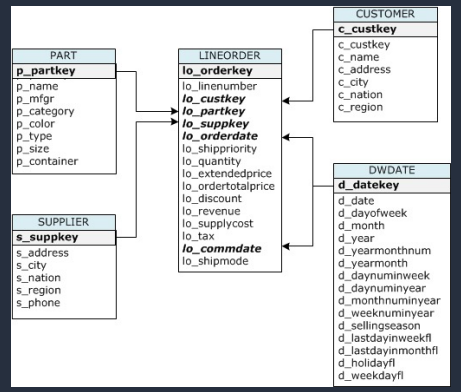
Data Modeling

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Redshift: Popular Data Models

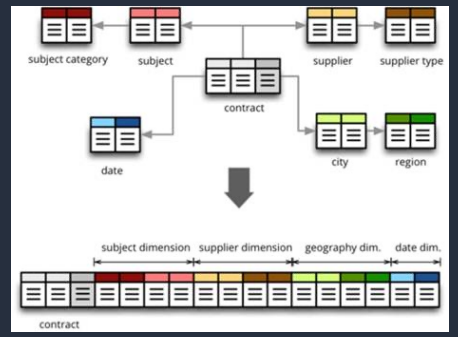
Redshift can be used with a number of data models including…

STAR

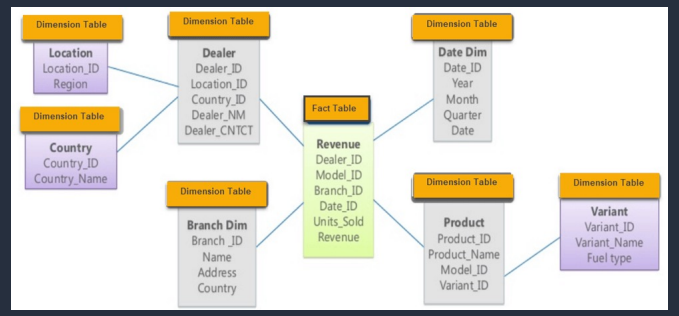


(models that are used less often)

Highly Denormalized



Snowflake



A commonly used data model with Amazon Redshift is the STAR schema, which separates data into large **fact tables** that reference any number of smaller **dimension (dim) tables**:

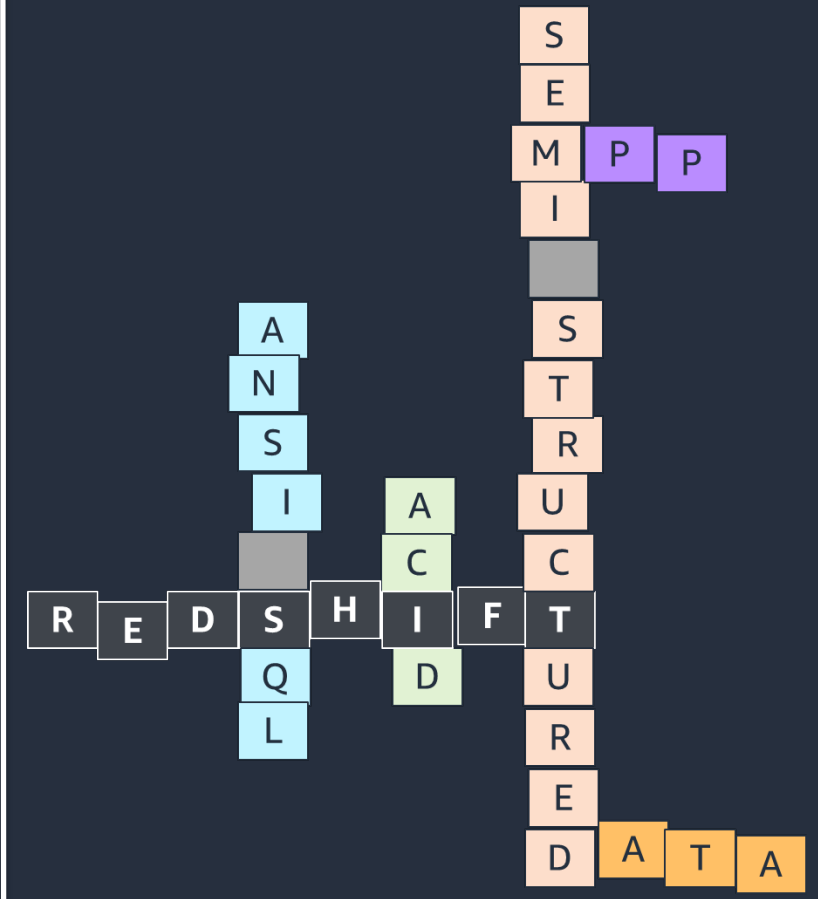
- Facts refer to specific events (e.g. order submitted.) and **fact tables** hold summary detail for those events. e.g. the high-level attributes of an order submitted such as *order_id*, *order_dt*, *product_id*, & *total_cost* are examples of columns that might be found in an *orders* fact table. Fact tables use foreign keys to link to dim tables
- The dimensions that make up a *fact* often have attributes themselves that are more efficiently stored in separate **dim tables**. e.g. a fact might contain a *product_id*, but the actual product details would be contained in a separate *products* dim table (e.g. *product_price*, *height_cm*, *width_cm*, & *product_id* are columns that might be found in a *products* dim table)

Note: The **3NF** data model resembles the STAR schema, but has much more table normalization and is typically more appropriate with OLTP systems

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Data modeling in Redshift: DB Basics

- Redshift is an ACID and ANSI SQL Compliant Data Warehouse
- Does not require indexes or db hints. Leverages sort keys, distribution, parallelism, and efficient data storage to achieve fast performance
- Parallelism, efficient columnar storage, and efficient I/O in Redshift are significantly impacted by choice of table distribution styles, data compression, and sort keys
- Redshift creates one database by default
- Use Schema quotas to apply governance for self service



© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Data modeling in Redshift: Data Basics

- Redshift tables and views are schema-on-write with support for schema-on-read via Redshift Spectrum
- Redshift organizes data in the following ways:
  - Database
  - Schemas
  - Tables and other DB Objects
- Support for scalar and schemaless Datatypes

Additional Documentation

- Ingesting and Querying semistructured data in Redshift
- Support for native Json and Semi Structured data
- **Supported data types in Redshift**

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Redshift Queries: 101

- Redshift is an ACID and ANSI SQL compliant cloud data warehouse that uses SQL syntax similar to **PostgreSQL syntax**
- Redshift supports **stored procedures (SPs)**
- Redshift supports custom **user defined scalar functions (UDFs)** using either a SQL `SELECT` clause or a Python program
- Redshift supports concurrent writes using **transactions** and serializable isolation

Redshift Query Editor

The Redshift Query Editor can be used to submit SQL queries for a Redshift database.

Among it's helpful features, is its real-time formatting of SQL as it is typed into the editor, visual summaries of table data, and visual query execution plans

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Redshift Stored Procedures: 101

- Stored Procedures (SPs) can incorporate DDL, DML and SELECT queries
- By default, **all users have create, drop, & execute privileges for SPs they own/create; superusers have the same on all SPs**
- **Security Definer allows the procedure to run with elevated privileges – take care to ensure the procedure can't be misused**
- SPs **don't need to return a value**, and can use procedural language, including **looping and conditional expressions**, to control logical flow

Redshift Stored Procedures (SPs)

Use the CREATE SQL command to create an SP. For example:

```
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;
```

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Useful tables to understand Query Behavior

System tables and views often hold information that can help uncover reasons for queries hanging, running longer than expected, etc.

- `STL_QUERY` - Returns execution information about a database query
- `STL_QUERY_METRICS` - Contains metrics information, such as the # of rows processed, CPU usage, input/output, and disk use, for queries that have completed running in user-defined query queues
- `STL_QUERYTEXT` - Captures the query text for SQL commands
- `STL_TR_CONFLICT`- Logs lock conflicts
- `STL_BCAST` - Logs information about network activity during execution of query steps that broadcast data

Additional Documentation

- *System tables and views*

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Choosing Data Types: Best Practices

- Redshift performance is about efficient I/O
- Make columns only as wide as they need to be
- Use appropriate types
- Multibyte Characters - Use VARCHAR data type for UTF-8 multibyte characters support (up to a maximum of four bytes)
- Spatial data can be natively stored, retrieved, and processed using the GEOMETRY data type and spatial functions.

Additional Documentation

- *Querying Spatial Data in Redshift*

```
create table sales(
listid varchar(255) not null,
sellerid varchar(255) not null,
buyerid varchar(255) not null,
eventid varchar(255) not null,
dateid varchar(255) not null,
qtysold varchar(255) not null,
pricepaid varchar(255) encode delta32k,
commission varchar(255) encode delta32k,
saletime varchar(255),
...
```

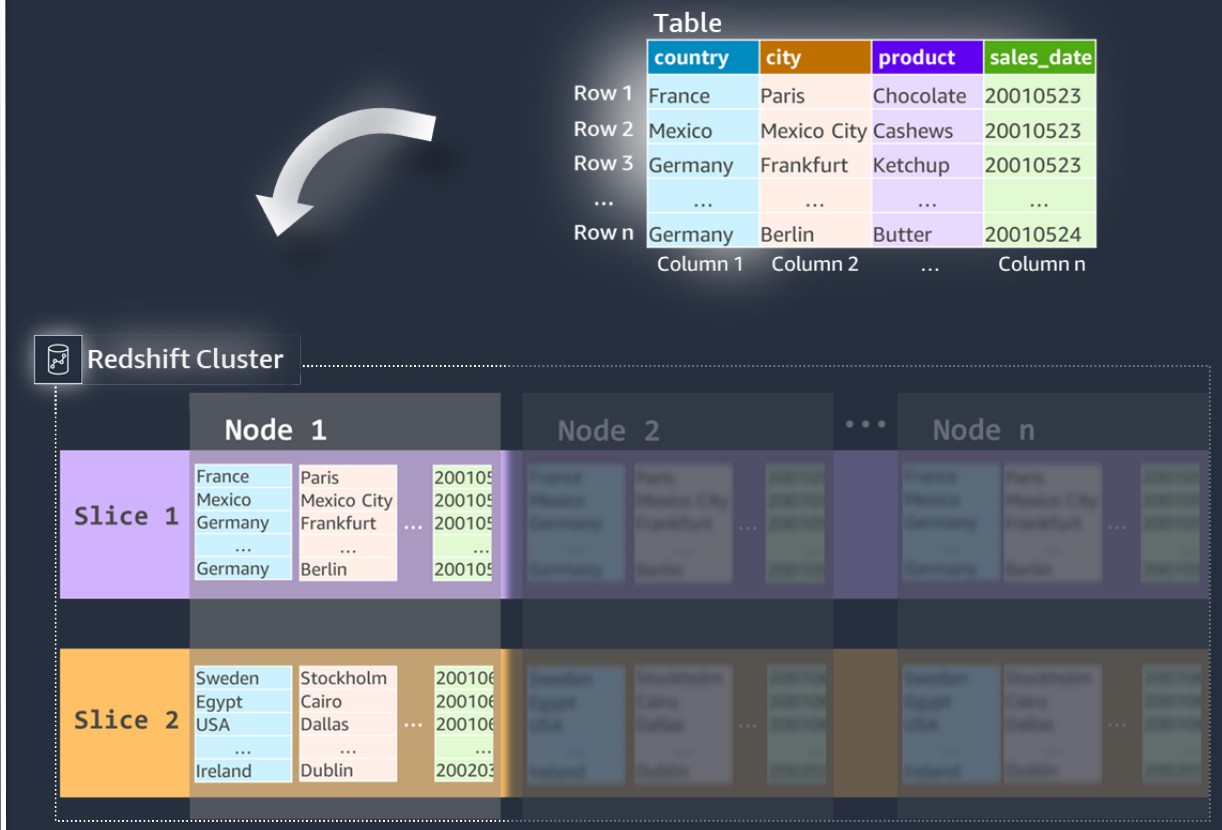
```
create table sales(
listid integer not null,
sellerid integer not null,
buyerid integer not null,
eventid integer not null,
dateid smallint not null,
qtysold smallint not null,
pricepaid decimal(8,2) encode delta32k,
commission decimal(8,2) encode delta32k,
saletime timestamp,
...
```

EFFICIENCY

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

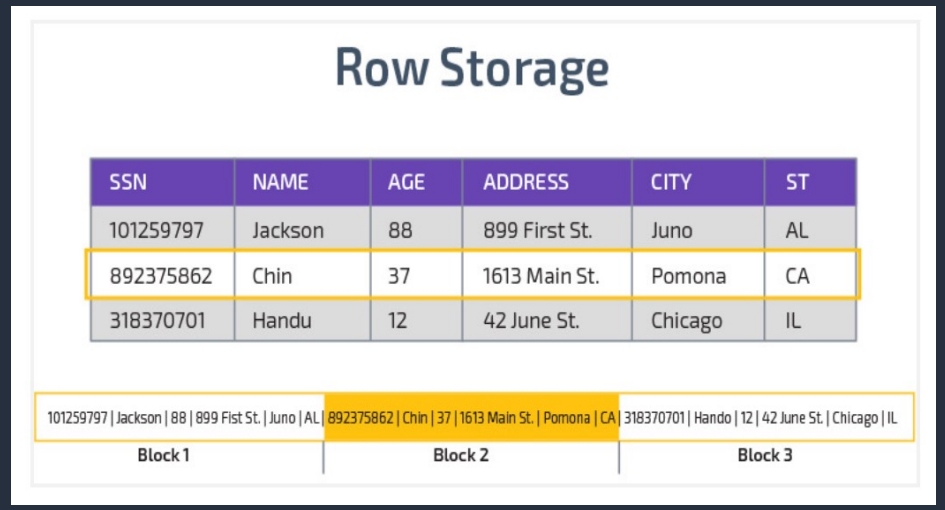
Redshift Storage: How Does it Work?

- Redshift is a columnar database, which means data on disk is physically organized by column
- Column data is stored to 1MB immutable blocks; a full block can contain as little as one value or as many as millions of values
- 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
- Each slice stores a set of blocks that contain a range of the values for each column

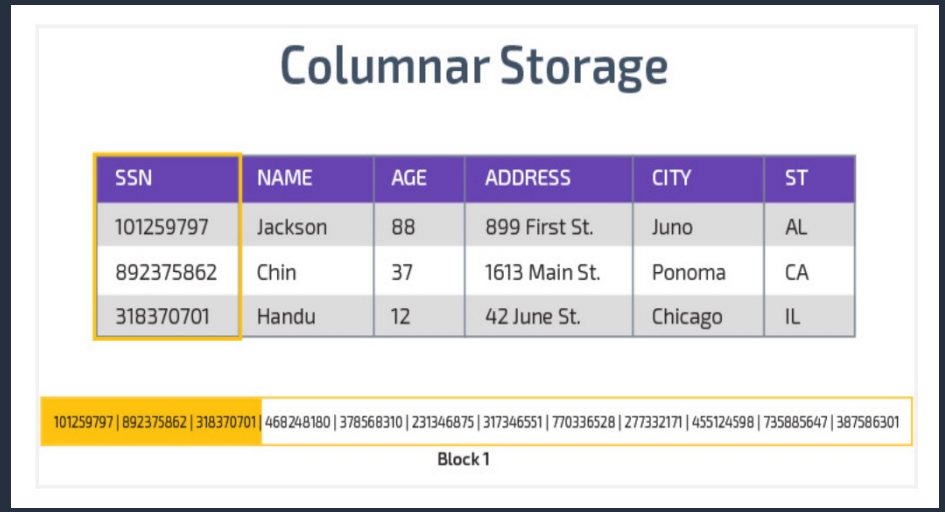


© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Row-Store vs Column Store



- Row storage (e.g. MySQL): all row fields are stored together on disk (typically in a sequential file)
- Accessing a column (example: scanning SSN of all residents) with row storage:
  - Scan every column in every row of the table
  - Resultant unnecessary I/O and caching overhead



- Column storage (e.g. Amazon Redshift): each table column is stored separately on disk (typically in a separate file or set of files)
- Accessing column (example: scanning SSN of all residents) with columnar storage:
  - Only scan blocks for relevant column(s)
  - Significant less I/O

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Row-Store Read vs Column Store Read

Given the following table definition and data for the `deep_dive` table, how will a simple SQL query behave in a row-based data store, and then in a column-based store?

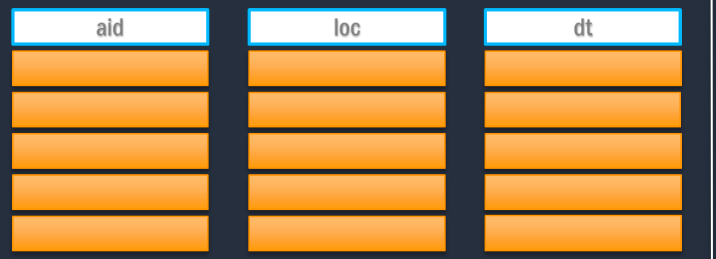
```
CREATE TABLE deep_dive (
     aid  INT        --airport_id
    ,loc  CHAR(3)    --location
    ,dt   DATE       --date
);
```

aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
3	SFO	2017-04-01
4	JFK	2017-05-14

```
SELECT min(dt) FROM deep_dive;
```

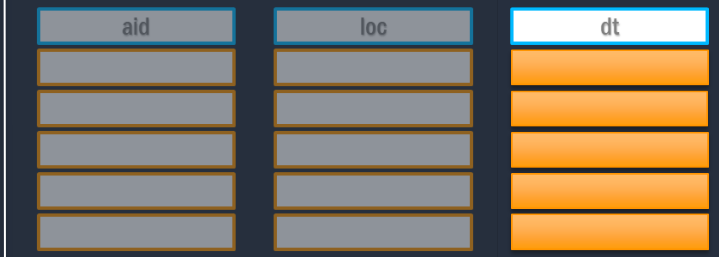
Row-based storage behavior

- Need to read everything
- Unnecessary I/O



Column-based storage behavior

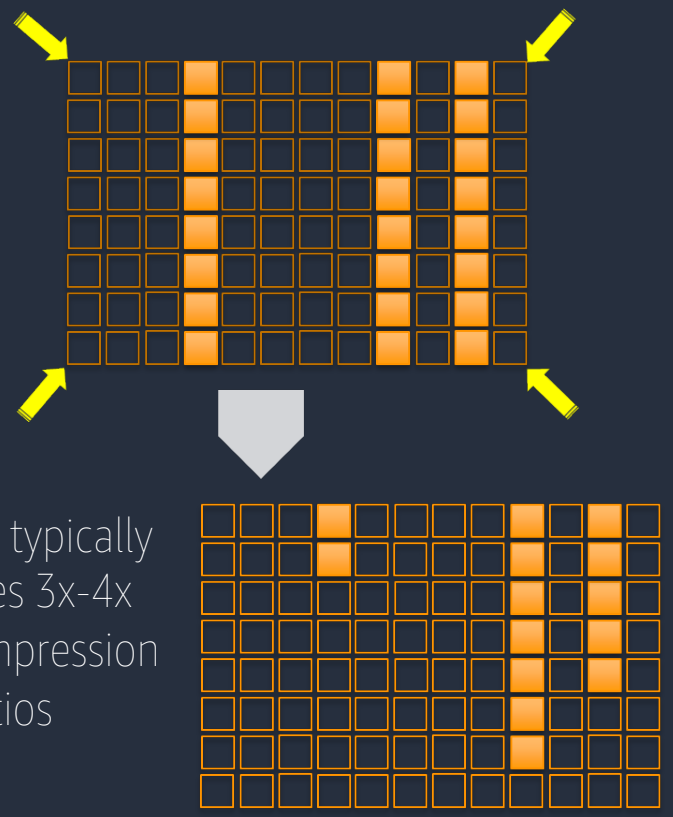
- Only scan blocks for relevant column
- Significantly less I/O



© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Columnar Compression

- Column-stores compress very efficiently
  - Each value in a single column is the same data type
  - Likely to have repeating values in a single column
- Redshift can typically achieve between 3x-4x data compression ratios
- Compression reduces storage requirements, but also improves performance by reducing I/O
- Columns grow and shrink independently in Redshift



Redshift typically achieves 3x-4x data compression ratios

Note:
In Redshift jargon, "Column Encoding" refers to "Compression"

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Redshift Distribution Styles

Distribution style is a table property that dictates how that table's data is distributed on internal storage

AUTO	Default (preferred) option; Redshift determines the best distribution style based on observed query interactions with tables, distribution style can change dynamically
KEY	A column value is hashed (usually a popular join column), and the same hash value is placed on the same slice
ALL	Full table data is placed on each compute node's first slice; recommended for relatively static tables < 3m rows
EVEN	Data is evenly distributed across all slices using a round-robin distribution

Data Distribution Goals

- Distribute data evenly for parallel processing
- Ensure each node has the same amount of data
- Minimize data movement during query processing

Additional Documentation

- *Automatic Table Optimization*
- *Distribution Styles and Keys*

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

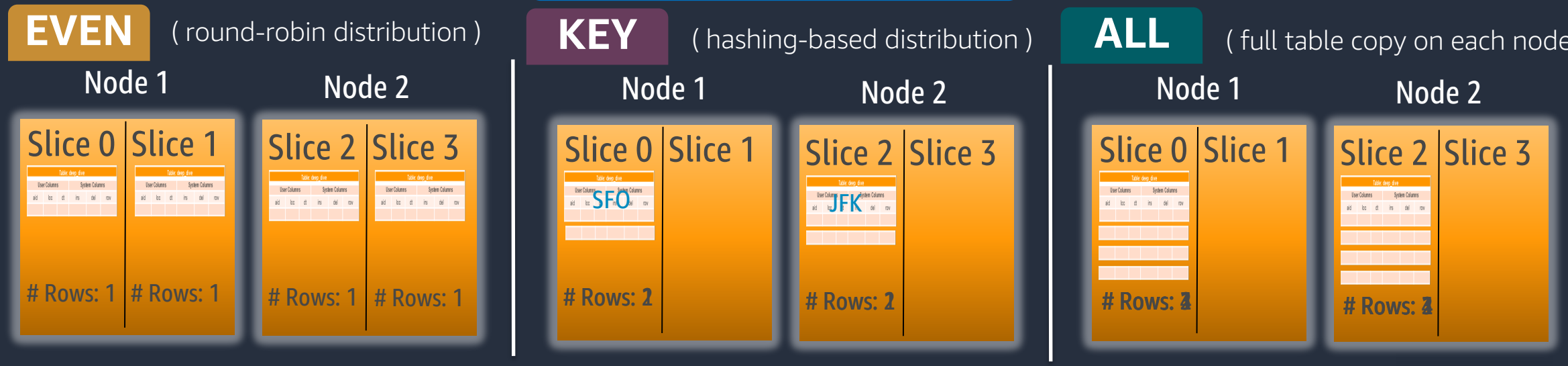
Data Distribution: Manual Examples

The `deep_dive` table has three columns; let's insert new rows to see the distribution styles in action for a two-node Redshift cluster, each node having two slices

```
CREATE TABLE deep_dive (
     aid     INT      --airport_id
    ,loc     CHAR(3)  --location
    ,dt      DATE     --date
) DISTSTYLE
   [EVEN | KEY DISTKEY (loc) | ALL];
```

```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```

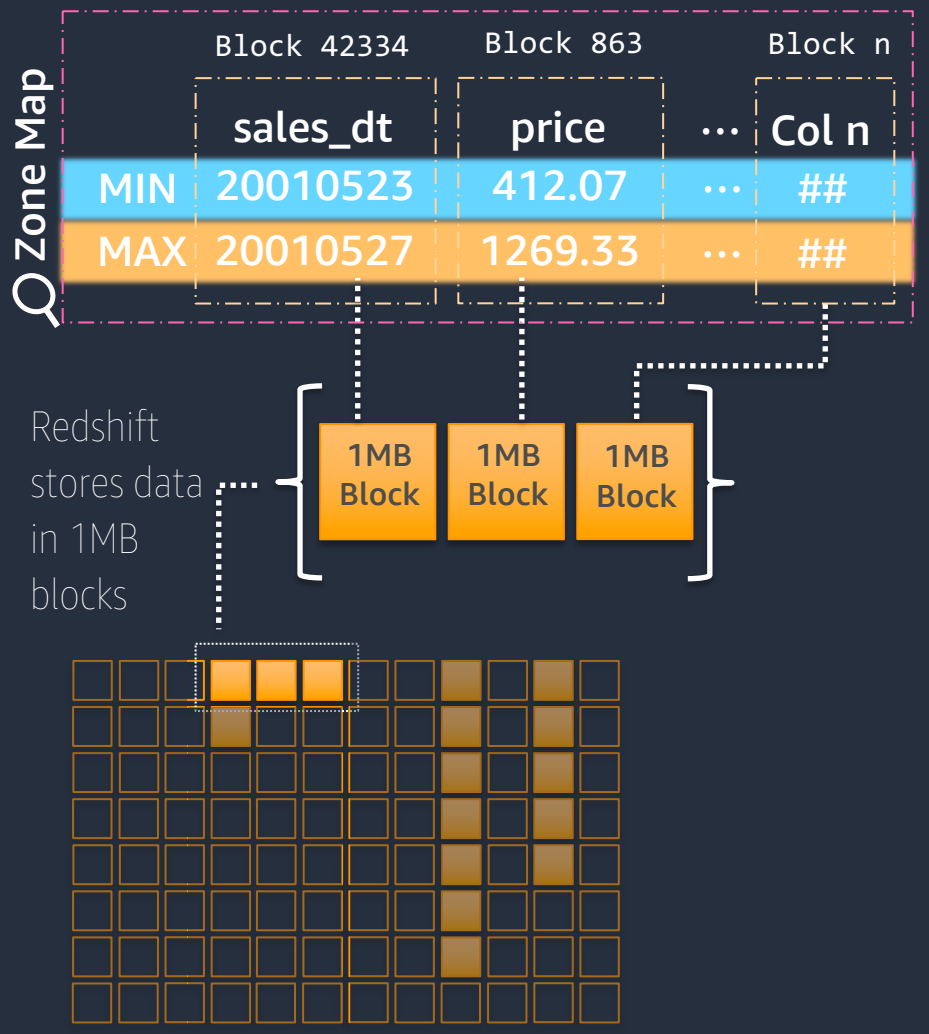
TIP: When using the *KEY* dist style, select a distribution key column that has **high cardinality** to help avoid row skew across slices



© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Redshift Sorting: Zone Maps

- Zone Maps are minimum-and-maximum values for each block of data
- Zone maps are stored in-memory and automatically generated
- Zone Maps allow Redshift to effectively prune blocks that cannot contain data needed for a given query
- Minimizes unnecessary I/O
- Along with sort keys, zone maps play a crucial role in enabling range-restricted scans to prune blocks and reduce I/O



© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Redshift Sort Keys

- Redshift uses sort keys to physically order data on disk
- Sort keys combined with *zone maps* function like an index in Redshift for a given set of columns and enable range-restricted scans to prune blocks and reduce I/O
- Compound Sort Key
  - Sort key is composed of one or more columns
  - Should be in order of lowest to highest cardinality
  - Most common use case - when the majority of table queries filter or join on a specific subset of columns

Redshift Sorting

When data is initially loaded into an 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.

Additional Documentation

- *Choosing the best sort key*
- *Compound and Interleaved sort keys*

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Zone Maps and Sorting: Example

```
CREATE TABLE deep_dive (
     aid    INT      --airport_id
    ,loc    CHAR(3) --location
    ,dt     DATE.    --date
) SORT KEY (dt, loc);
```

```
SELECT count(*)
FROM deep_dive
WHERE dt = '06-09-2017';
```

Zone maps and sort keys can serve as a significant optimization by reducing the number of blocks examined (and therefore I/O) during query execution

Unsorted table

MIN: 01-JUNE-2017
MAX: 20-JUNE-2017

MIN: 08-JUNE-2017
MAX: 30-JUNE-2017

MIN: 12-JUNE-2017
MAX: 20-JUNE-2017

MIN: 02-JUNE-2017
MAX: 25-JUNE-2017

Sorted by date

MIN: 01-JUNE-2017
MAX: 06-JUNE-2017

MIN: 07-JUNE-2017
MAX: 12-JUNE-2017

MIN: 13-JUNE-2017
MAX: 21-JUNE-2017

MIN: 21-JUNE-2017
MAX: 30-JUNE-2017

Additional Documentation
- AWS Table Design

Relevant Blogs
- Automatic Table Optimization
- Advanced Table Design Playbook

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

ALTER Table Command

In addition to tasks such as renaming a column's name or changing its data type, the ALTER command can be used to change a table's distribution style and/or key

- Change a table's distribution key

```
ALTER TABLE table_name
    ALTER DISTKEY column_name
```

- Change a table's distribution style

```
ALTER TABLE table_name
    ALTER DISTSTYLE
[ALL | EVEN | KEY DISTKEY column_name | AUTO]
```

Additional Notes on Altering an Existing Table

- The ALTER table operation locks the table for read and write operations until the operation completes
- For tables with a SORTKEY defined, it is recommended to run a VACUUM [FULL|SORT ONLY] command (to re-sort the table), after ALTER DISTKEY runs

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Alter SORTKEY

```
ALTER [COMPOUND] SORTKEY (column_name
[,...])
```

- Use to change or add the compound sort key used for a table
- 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

Redshift Sorting

Table compound sort keys are not static – they can be altered after table creation using the **ALTER** command

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Materialized Views

- Materialized view runs the associated SQL query and persists the result set
- Materialized views can be refreshed automatically or manually
- Redshift automatically determines best way to update data in the materialized view (incremental or full refresh)
- Automatic query rewrite leverages relevant materialized views and can improve query performance by order(s) of magnitude
- Benefits – Improved Performance
  - Optimize complex, long-running, resource-intensive queries on large tables
  - When materialized view is queried, Redshift returns precomputed results without having to access the base tables
  - Especially useful for predictable and repeated queries

Redshift Materialized Views

Materialized views can be created using the **CREATE** statement, and can be included (default) or excluded from Redshift backups. Materialized views can also have table attributes such as *dist style* and *sort keys*, and be refreshed at any time

```
CREATE MATERIALIZED VIEW mv_name
[ BACKUP { YES | NO } ]
[ table_attributes ]
AS query

REFRESH MATERIALIZED VIEW mv_name;
```

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Materialized Views Auto Refresh and Query Rewrite

- Provide significantly faster query performance for repeated and predictable analytical workloads, such as ETL, BI/dashboarding queries
- Materialized Views can be based on one or more Amazon Redshift tables or external tables (Spectrum, Federated)
- Amazon Redshift auto query rewrite optimizes queries by replacing native tables with materialized views
- Simplify usage of up-to-date materialized views using a scheduled, automatic or manually timed refresh

For refresh status, check SVL_MV_REFRESH_STATUS,

Only up-to-date (fresh) materialized views are considered for automatic query rewrites

Use STV_MV_INFO table to check if materialized views are eligible for automatic query rewrites

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

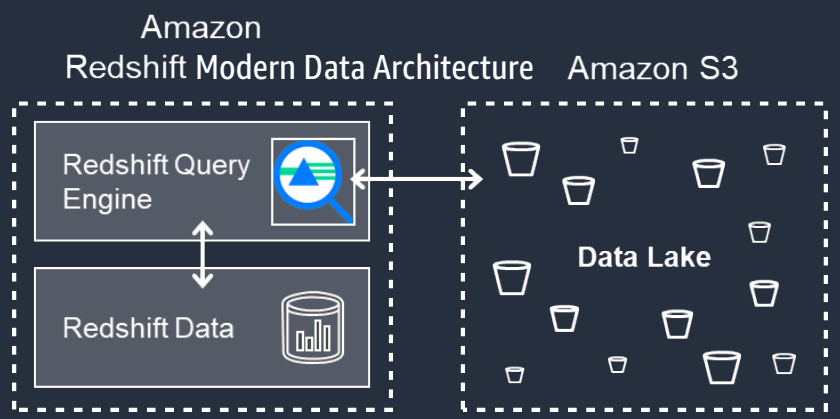
Data modelling for Data Lakes and Redshift Spectrum

- With Data Lakes, tables are collections of files
- Data Lake file types have a huge influence on performance of Redshift Spectrum queries
- Best Practices
  - Number of files in Data Lake should be a multiple of your Redshift slice count (general best practice)

    Redshift Spectrum can automatically split Parquet, ORC, text-format, and Bz2 files for optimal processing
  - File sizes should be in the range 64MB – 512MB
  - Files should be of uniform size (especially for files that can't be automatically split by Redshift Spectrum such as Avro and Gzip) to avoid execution skew; Spectrum queries only run as fast as the slowest node in the cluster

Redshift/Data Lake Interactions

Redshift Spectrum is a feature of Redshift that enables queries to reference external tables in your S3 data lake



© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Data Lake Partitions

- Spectrum will perform 'partition pruning' based on structure of data on S3
  - This is actually based on Hive metastore pruning, rather than Redshift Spectrum specific - also works in EMR, Glue, etc.
  - Uses Glue Data Catalog Partition columns

- Given a table in `s3_bucket/prefix` `s3://mybucket/prefix/mytable:`
  - Partitions are based on S3 prefix
  - Tables may have thousands of partitions

Redshift/Data Lake Interactions

Multiple date partition columns have fixed inclusion order. For example:

Partition:

```
s3://mybucket/data/marketing/impres
sions/application=flux_capacitor/yy
yy=<year>/mm=<month>/dd=<day>
```

The query below will not use partition pruning (year not provided)

```
select sum(hits) from
marketing_impressions where
application_name = 'flux_capacitor'
and mm='02‘
```

Need to add "`and yyyy='2018'`"

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Predicate pushdown to improve Query Performance

Advanced

- Some types of SQL operations can be evaluated completely within the Spectrum layer
- Examples of Spectrum-layer operations
  - `GROUP BY` clauses
  - Comparison & pattern-matching conditions (e.g. `LIKE`)
  - Aggregate functions (e.g. `COUNT`, `SUM`, `AVG`, `MIN`, & `MAX`)
  - String functions
- Operations that can't be pushed to the Redshift Spectrum layer include `DISTINCT` and `ORDER BY`

  Perform `ORDER BY` only at the top level of the query if possible, since sorting is done in the leader node
- Use `SVL_S3QUERY_SUMMARY` system view to evaluate predicate pushdown efficiency

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Data Loads and Unloads

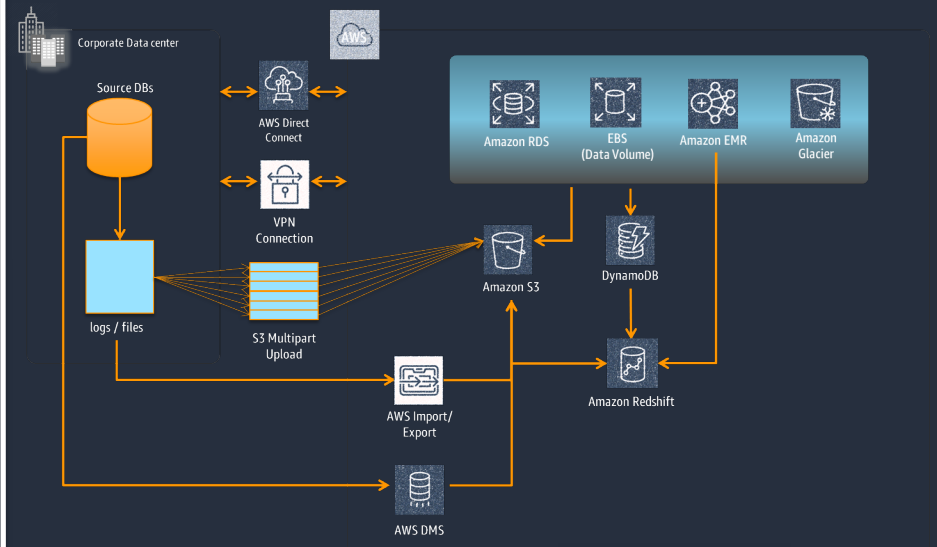
© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Data Loading Methods

- Some Redshift data loading operations use multiple transactions, while others use a single transaction
- Data loading that traverses the leader node each time and requires separate transactions
  - `INSERT`, `INSERT(multi-value)`, `UPDATE`, `DELETE`
- Data loading that engages leader node once, uses a single transaction, and engages compute nodes in parallel
  - `COPY`, `UNLOAD`, `CTAS`, `CREATE TABLE (LIKE...)`, `INSERT INTO (SELECT...)`
  - Loading data from S3, Dynamo DB, EMR, and SSH Commands

Data Loading

A number of options exist for loading data into Amazon Redshift

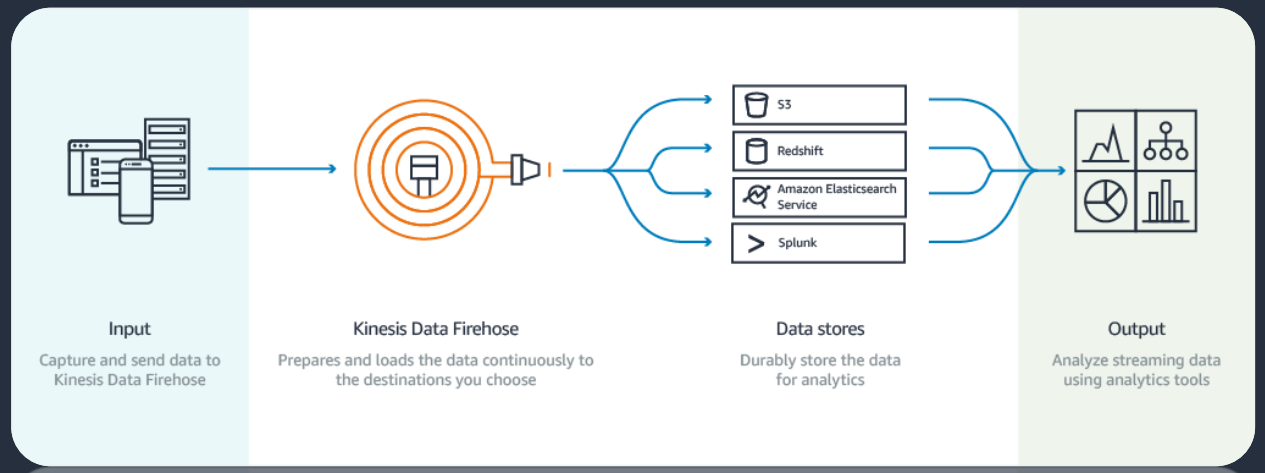


© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Loading Data from Kinesis Firehose

Advanced

- Kinesis Firehose can automatically load streaming data to Amazon S3, Redshift, and ElasticSearch
- Firehose micro-batches streaming messages and persists to S3 and then loads to Redshift via COPY command with manifest file
- Specify batch size in MB to load (1-100MB) & the # of seconds before a load occurs (60-900 sec)
- Multi-AZ buffering of data prior to load
- Firehose manages all compression & encryption
- Automatic load retry



© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Data Loading: COPY Command

Additional Documentation

- Copy Command - more info

- General Syntax

e.g. S3 object name or folder path

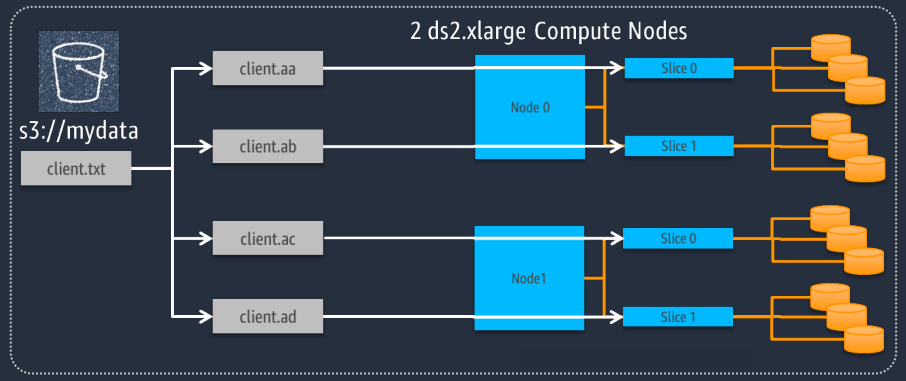
```
COPY <TABLE> from <location>
credentials "aws_access_key_id=<access_key>;
aws_secret_access_key=<secret_key>"
iam_role "arn"
region;
```

- A number of COPY options can be specified, including
  - CSV, JSON, or Avro Data
  - Supports columnar formats – Parquet, ORC
  - Compression Options
  - Encryption Options
  - Transformation Options
  - Date formats through DATEFORMAT, TIMEFORMAT
  - ACCEPTINVCHAR to replace invalid chars with '?'
  - COMPUPDATE PRESET/ON ensures optimal column encodings

Data Loading

The COPY command typically outperforms single-row inserts, because it engages compute nodes to load data in parallel

```
copy customer from 's3://mydata/'
credentials 'aws_access_key_id=<your-access-key>;
aws_secret_access_key=<your_secret_key>'
delimiter '|';
```



© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

COPY Command: Fine-grained access control

Advanced

- Create fine-grained access control policy to allow specific users and groups to run COPY command.
- With `AssumeRole` privilege, you can restrict the users and groups who can run the COPY command.
- Example - grants the `ASSUMEROLE` privilege to the user `user1` for the IAM role `Redshift-S3-Read` to perform COPY operation.

```
grant assumerole on 'arn:aws:iam::<accountid>:role/Redshift-S3-Read’ to
user1 for copy;
```

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Data Loading: Best Practices (Technique)

- Use `COPY` command to load data whenever possible
- Use a single `COPY` command per table
  - Writes are serial per table
  - Commits are serial per cluster
- Use multi-row inserts if `COPY` is not possible
  - Bulk insert operations (`INSERT INTO...SELECT` and `CREATE TABLE AS`) provide high performance data insertion
- Use staging tables to perform an upsert

Additional Documentation

- *Loading from S3*

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

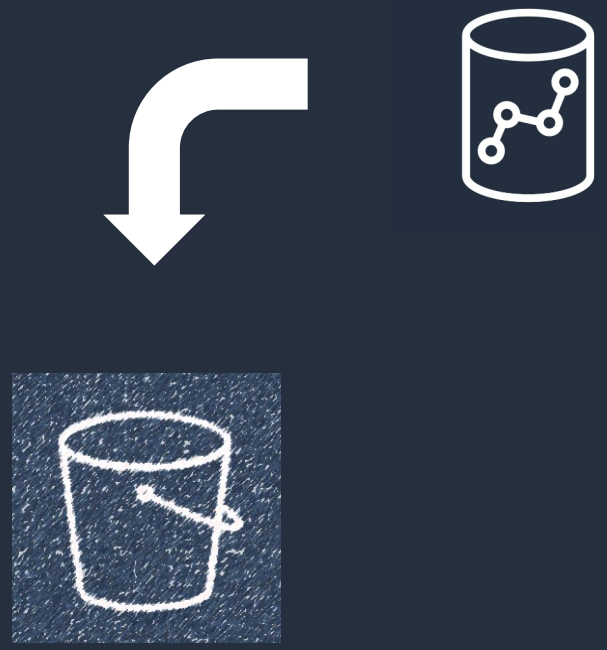
Data Loading: Best Practices (Post-Load)

- Redshift automatically performs vacuum and analyze in the background during periods of low workloads.
- Redshift users are still empowered to explicitly invoke **VACUUM** and then **ANALYZE** as part of their workloads
  - Explicitly invoking VACUUM and then ANALYZE ensures that a table is sorted, defragmented, and analyzed immediately and with priority for the benefit of the next steps in a workflow.

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Unloading Data: UNLOAD Command

- `UNLOAD` command is the reverse of `COPY`, in that it outputs data from Amazon Redshift to S3
  - Runs from a `SELECT` statement. `Order By` clause respected by `UNLOAD` if `PARALLEL=OFF`
  - Encryption & compression handled automatically
  - Runs in parallel on all compute nodes
- `UNLOAD` output
  - CSV or Parquet (**Data Lake Export**) file formats
  - Generates > 1 file per slice for all compute nodes
  - Max file size written on S3 can be controlled (max internal limit 6.2GB)
  - Generates a manifest for all unloaded files (useful for `COPY` into another cluster)
  - Control if files can overwrite existing locations or not



```
UNLOAD ('select-statement')
TO 's3://object-path/name-prefix'
iam_role “arn” [ option [ ... ] ]
```

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

UNLOAD Command: Fine-grained access control

- Create fine-grained access control policy to allow specific users and groups to run UNLOAD command.
- With `AssumeRole` privilege, you can restrict the users and groups who can run the UNLOAD command.
- Example - grants the `ASSUMEROLE` privilege to the user `user1` for the IAM role `Redshift-S3-Read` to perform UNLOAD operation.

```
grant assumerole on 'arn:aws:iam::<accountid>:role/Redshift-S3-Read’ to user1 for unload;
```

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

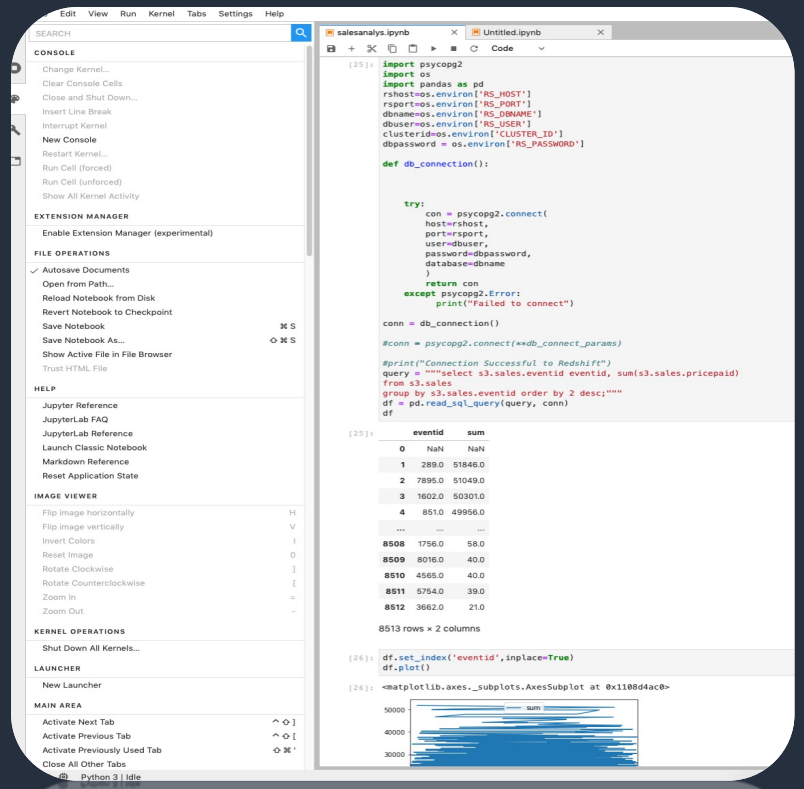
Seamless Integration with AWS Services

Migration | ETL | Machine Learning | Data Visualization

- AWS Database Migration Service (DMS)
  Fully-managed migration service
- AWS Glue
  Fully managed extract, transform, and load (ETL) service
- Amazon EMR
  Cloud-native big data platform
- Amazon Sagemaker
  Fully managed service to build, train, and deploy machine learning (ML) models
- Amazon Quicksight
  Fast, cloud-powered business intelligence service

and more…

Sagemaker / Jupyter Notebook



© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Query Tuning

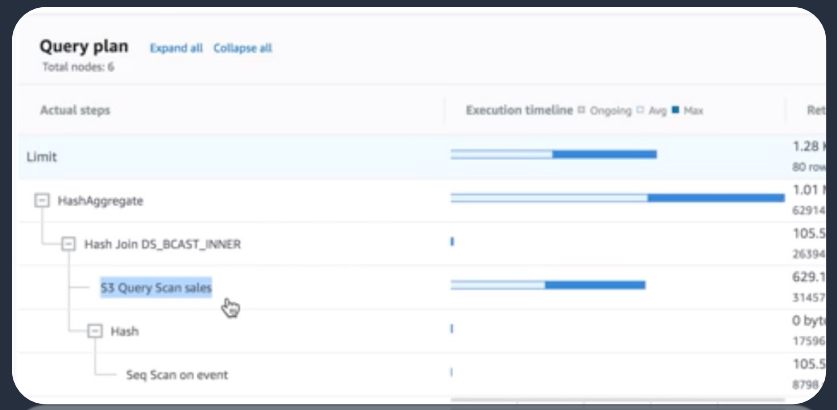
© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Explain-Plan: Basics

- Use the `EXPLAIN` command to view the explain-plan for a query
  - Syntax: `EXPLAIN <SELECT|UPDATE|etc. ...>`
  - Generates pre-formatted output with one row per step and a single column indicating the step
  - Read explain-plans from the bottom-up
- Redshift Query Editor provides a visual execution plan for a query
- Explain-plans explain a lot, including…
  - Execution order of steps for local & Spectrum portions
  - Estimated number of rows to be returned by a step
  - Amount of data to be sent across slices per step (network I/O is expensive & should be optimized as much as possible)
  - Relative cost of a step

Redshift Query Editor

The Redshift Query Editor offers a visual execution plan for queries that intuitively shows how work was distributed for a query



© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Explain-Plan: Example

Analyst's question: What is the average wait time for a listing to be sold?

```
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)
```

```
EXPLAIN
  SELECT AVG(DATEDIFF(day, listtime, saletime)) as avgwait
  FROM sales, listing
  WHERE sales.listid = listing.listid;
```

Relative cost of operation (cumulative)

Estimated avg width of rows (bytes)

No data distribution across cluster nodes required

Data Distribution: Manual Examples

```
                                    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)
```

Aggregate results

where clause condition

where clause

Table scan (listing)

Table scan (sales)

Estimated # of rows to be returned

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Explain-Plan: Network Transmission

Advanced

During join steps, a slice may need to work with data not stored locally, and network transmission is by far the most expensive operation a query will do

DS_DIST_NONE	This is the ideal situation which indicates that the data for the join is co-located on the same slice; this is the most efficient option as no network transfer will occur
DS_DIST_ALL_NONE	Indicates that the join is occurring with a table that has `DISTSTYLE ALL` and also does not incur network transfer
DS_DIST_ALL_INNER	Indicates that the 'inner' join table is being sent to a single node because the join table uses `DISTSTYLE ALL`; this join is executed on a single node and will likely be very slow
DS_DIST_INNER (|OUTER)	Indicates which table is being redistributed when using an outer join (inner or outer table); if one of the tables is much smaller or infrequently updated, consider changing it to `DISTSTYLE ALL`
DS_BCAST_INNER	Indicates that the 'inner' join table is being broadcast to all nodes
DS_BCAST_BOTH	Indicates that both tables in the join are being broadcast to all nodes. This is the worst possible option

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Execution-Plan: Visual example

The Redshift Query Editor offers a visual execution plan for queries

edshiftv2/home?region=us-east-1#query-editor:

dpanda @ 2653-6122-8846 | N. Virginia | Support

```
ORDER BY 2 DESC;

SELECT  top 10 TOTAL_SALES.Eventid, sum(pricepaid)
FROM TOTAL_sales, event
WHERE TOTAL_sales.Eventid = event.Eventid
GROUP BY TOTAL_sales.Eventid
ORDER BY 2 DESC;
```

Run | Save | Clear | Send feedback

Query history | Query results | Table details

Query 641407

Execution | Data | Visualize

Completed, started on November 16, 2019 at 20:32:36

ELAPSED TIME: 00 m 02 s

Query plan Expand all Collapse all

Total nodes: 6

Show all plan nodes | Text plan | Execution timeline | Execution time

Actual steps	Execution timeline (Ongoing, Avg, Max)	Returned	Cost
Limit		1.28 KB, 80 rows	1001230767618.65, 100%
HashAggregate		1.01 MB, 62914 rows	1230767610.98, 0%
Hash Join DS_BCAST_INNER		105.58 KB, 26394 rows	1230766610.48, 0%
S3 Query Scan sales		629.14 KB, 31457 rows	175002000.5, 0%
Hash		0 bytes, 17596 rows	87.98, 0%
Seq Scan on event		105.58 KB, 8798 rows	87.98, 0%

Completed

From November 16, 2019 at 08:32:35 PM

To November 16, 2019 at 08:32:35 PM

Statistics

Rows returned	379,509
Total data scanned	4.37 MB

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Explain-Plan: Source of Historic Trends

- Use STL_EXPLAIN table to get explain plans for queries previously run on the system
- Allows you to analyze historic trends

```
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=6350.30... |
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 |
```

Redshift Query Tuning

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

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

How to Terminate a Query?

- Use CANCEL command to cancel the query

```
set query_group to 'superuser'; --optional
CANCEL <pid>;
reset query_group; --optional
```

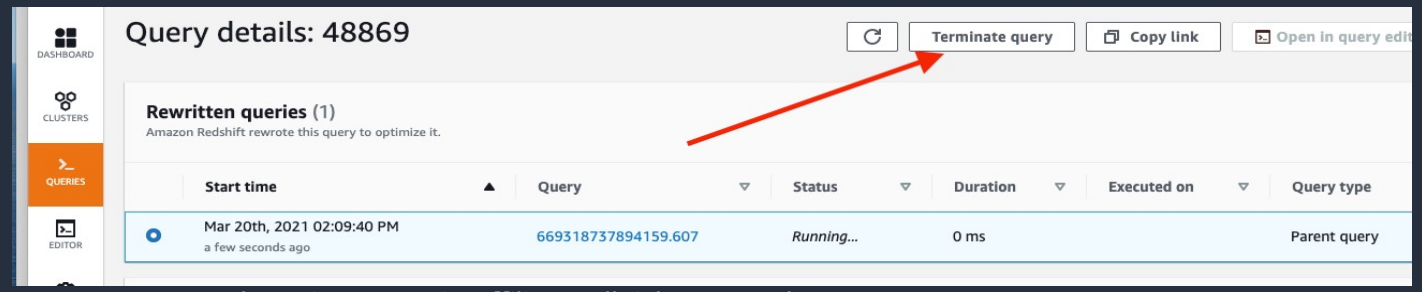
- Then, if the query was in a transaction – use the ABORT or ROLLBACK command to abort that transaction

```
ABORT;
```

- To force-kill a query/session, use pg_terminate_backend

```
SELECT * FROM SVV_TRANSACTIONS; --get session PID
SELECT pg_terminate_backend(<pid>);
```

- Alternately use the console



Redshift Query Tuning

To get a query's PID, you can execute a query similar to the following:

```
SELECT pid, trim(user_name),
starttime, substring(query,1,20)
FROM STV_RECENTS
WHERE status='Running';
```

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Query Best-Practices: 101

- Good queries are those that are simple to maintain over time, and which are intuitive when read
- Settle on a syntax style as a team (ANSI or Equijoin)
  - ANSI: `from <table> a inner join <table> b on a.id = b.id`
  - Equijoin: `from <table> a, <table> b where a.id = b.id`
- Filter whenever possible

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Table Defragmentation/Organization

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Redshift Table Storage: 101

- Redshift uses 'append only' data storage
- Data blocks in column-storage files are never modified, new blocks are added to the end of the file.
- Deleted/updated rows:
  - Represent wasted storage that could be used for business data
  - Cause fragmentation that negatively impacts performance
  - Could cause columns to lose their "sortedness"
- Redshift automatically cleans up wasted space with Auto-Vacuum.
- Use VACUUM command to increase available storage and improve performance.

Redshift Table Storage

Use time-series approach to avoid large, recurring merge operations that could result in a fragmented or unsorted table.

Multiple tables with identical schemas each hold information that pertains to a specific date-range (e.g. daily, weekly, monthly, etc.)

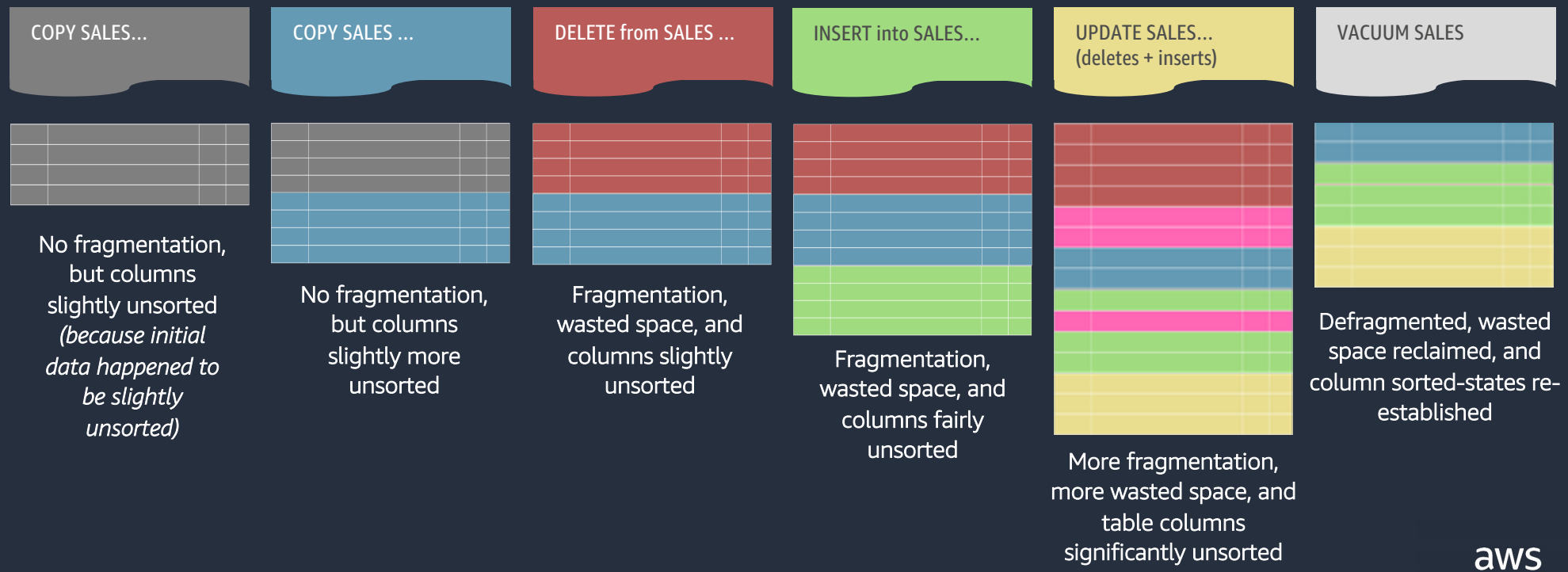
Additional Documentation: Vacuuming Tables

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Defragmentation/Organization: Example

```
CREATE TABLE SALES (
    col1      INT
   ,col2      CHAR(3)
   ,...
   ,coln      DATE
);
```

Let's see how loading, deleting, inserting, and updating rows impacts the storage-efficiency and organization of the SALES table, and then finally the impact of the VACUUM command on the SALES table



© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Table Defragmentation/Organization: Auto-Vacuum

- *Auto-Vacuum* monitors changes to your workload and automatically reclaims disk space occupied by rows affected by UPDATE and DELETE operations
- Restores the sorted state of data (lessens the need to run vacuum sort)
- Enabled by default, & runs every hour during periods of light workloads
- Automatically pauses if cluster workloads spike and resumes when workloads decrease
- Use Redshift Advisor to identify fragmented/unsorted tables

Redshift Auto-Vacuum-Delete

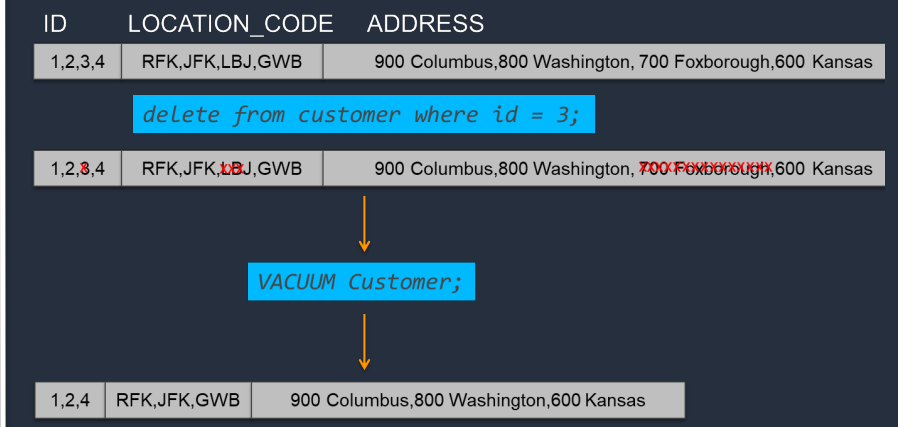
Can be tracked by monitoring *'Space reclaimed by auto vacuum delete'* on the Cluster Performance tab on the Redshift Console and using the CloudWatch metric *AutoVacuumSpaceFreed*

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

VACUUM Command: Options

VACUUM [FULL | SORT ONLY | DELETE ONLY | REINDEX] [[table_name] [TO threshold PERCENT]]

- VACUUM FULL [TABLE_NAME]
  Full vacuum reclaims all space from DELETE or UPDATE operations; skips sort phase for any tables > 95% sorted
- VACUUM SORT ONLY [TABLE_NAME]
  Resorts the table but does not reclaim table free space which makes it faster than VACUUM FULL. Useful when data has not updated or deleted significantly, but is not inserted in sort key order. Skips any tables that are > 95% sorted
- VACUUM DELETE ONLY [TABLE_NAME]
  Reclaims free space from the table but does not re-sort data on disk. Useful when data is inserted in sort key order
- VACUUM REINDEX [TABLE_NAME]
  Only used for tables with *Interleaved Sort Keys*; sorts data by interleaved z-curve



Additional Documentation: Vacuum Command

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Table Statistics

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Redshift Table Statistics: 101

- Table and column statistics allow Query Plan Optimizer to create a more performant plan
- Table statistics
  - Number of rows/bytes and discrete values in each column
  - Skew of values
  - Help determine optimal order of filters and joins
- Statistics are automatically updated in the background by default with *Auto-Analyze*
- External tables (i.e. Redshift Spectrum tables) cannot be analyzed.
    - Glue crawler does update numRows which will help performance

Redshift Operations

Amazon Redshift automatically collects table statistics after row updates and deletes. You may also, a user or explicitly use ANALYZE command update a table's statistics immediately and with priority

Additional Documentation: Analyzing tables

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Redshift Table Statistics: Auto-Analyze

- *Auto-Analyze* monitors changes to your workload and automatically updates statistics in the background
- Enabled by default, and runs every hour during periods of light workloads
- Delays start if user queries are running or have been running for more than 15 seconds in the last 300 secs
- Automatically pauses if cluster workloads spike and resumes when workloads decrease
- Processes tables in windows of 100M rows
- Skips tables that already have up-to-date statistics

Redshift ANALYZE Command

ANALYZE command will skip tables that already have up-to-date statistics or where the extent of changes to the table data is small

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

How to Update Redshift Table Statistics?

- COPY command updates statistics automatically when loading an empty table (unless STATUPDATE OFF is specified)
- Statistics are automatically updated when certain tables are first created
  - CREATE [TEMP] TABLE AS SELECT
  - SELECT INTO
- ANALYZE command can collect statistics for a column, table, or the entire database
  - ANALYZE ; *(Analyzes the entire database)*
  - ANALYZE [TABLE_NAME] ; *(Analyzes a single table)*
  - ANALYZE [TABLE_NAME](col1, col2) ; *(Analyzes specific columns only)*

Redshift ANALYZE Command

ANALYZE is a relatively inexpensive operation that can be run during maintenance windows or during normal business hours.

Additional Documentation:
- *Analyze command*
- *Analyze command history*

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

ANALYZE Best Practices

- ANALZYE a table after
  - Modifying/loading large # of rows (e.g. > 5% of existing rows)
  - Changing distribution or sort keys
  - Adding or removing foreign keys
  - Vacuum operation
- ANALYZE the columns that are frequently used in
  - Sorting and grouping operations
  - Joins
  - Query Predicates
- Do not ANALYZE all tables frequently; statistics do not expire with time; no data change = no statistics change!
- Use Redshift Advisor to identify stale table statistics

Redshift ANALYZE Command

ANALYZE will skip tables that are greater than 97% sorted.
If ANALYZE completed faster than you thought, this could be the reason.

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Thank You!

Gartner Magic Quadrant 2021

AWS named a Leader in the Cloud Database Management Systems Magic Quadrant with the highest score in Ability to Execute among the 16 vendors evaluated

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.