



Splice Machine Documentation

Last generated: March 12, 2018

© 2018 Splice Machine, Inc. All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.

Table of Contents

Splice Machine Developer Tutorials

Splice Machine Tutorials

Introduction	1
--------------------	---

splice> Command Line

Introduction	4
Getting Started With splice>	5
Scripting splice> Commands	18
Using RLWrap	23

Configuring Security

Introduction	25
Roles and Authorization	26
Configuring Authentication	29
Configure SSL/TLS	34
Using HAProxy with Kerberos	42

Importing Data

Introduction	45
Overview of Importing Data	46
Import Parameters	52
Handling Import Input Data	64
Import Error Handling	74
Examples of Importing Data	76
Examples of Bulk HFile Import	82
Importing TPCH Benchmark Data	93

Accessing Data in the Cloud

Introduction	118
Setting up S3 Access	119
Uploading Data to S3	127

Connecting with JDBC

Introduction	135
Connecting via JDBC with Java	137
Connecting via JDBC with JRuby	140
Connecting via JDBC with Jython	143

Connecting via JDBC with Python	146
Connecting via JDBC with R	147
Connecting via JDBC with Scala	149
Connecting via JDBC with Angular/NodeJS	153

Connecting with ODBC

Introduction	154
Installing our ODBC Driver.....	155
Connecting via ODBC with Python	171
Connecting via ODBC with C	174

Streaming Data

Introduction	176
Using Apache Storm	177
MQTT Spark Streaming	181
Creating a Kafka Producer.....	190
Configuring a Kafka Feed	191

Connecting BI Tools

Introduction	192
Connecting Cognos	193
Connecting DBeaver.....	194
Connecting DBVisualizer	202
Connecting SQuirreL	206
Connecting Tableau	214

Analytics and Machine Learning

Introduction	216
Using Zeppelin	217

Splice Machine SQL Reference Manual

Splice Machine SQL Manual

Introduction	222
SQL Summary.....	223
SQL Limitations.....	225
SQL Reserved Words	230
SQL Argument Matching.....	240

Identifiers

Introduction	243
--------------------	-----

Identifier Types.....	245
Identifier Syntax	248

Data Types

Introduction	250
BIGINT	254
BLOB.....	255
BOOLEAN.....	259
CHAR.....	260
CLOB.....	262
DATE.....	264
DECIMAL.....	266
DOUBLE	268
DOUBLE PRECISION.....	270
FLOAT.....	272
INTEGER.....	273
LONG VARCHAR.....	275
NUMERIC	276
REAL.....	278
SMALLINT.....	280
TEXT	281
TIME.....	283
TIMESTAMP	285
VARCHAR.....	288

Statements

Introduction	289
ALTER TABLE.....	293
CALL (Procedure).....	300
column-definition.....	301
CREATE EXTERNAL TABLE.....	303
CREATE FUNCTION	307
CREATE INDEX.....	312
CREATE PROCEDURE.....	316
CREATE ROLE	319
CREATE SCHEMA	321
CREATE SEQUENCE.....	323
CREATE SYNONYM.....	326
CREATE TABLE	328
CREATE TEMPORARY TABLE.....	333

CREATE TRIGGER	336
CREATE VIEW	340
DECLARE GLOBAL TEMPORARY TABLE	343
DELETE	345
Dependencies	347
DROP FUNCTION	349
DROP INDEX	350
DROP PROCEDURE	351
DROP ROLE	353
DROP SCHEMA	354
DROP SEQUENCE	355
DROP SYNONYM	356
DROP TABLE	357
DROP TRIGGER	358
DROP VIEW	359
generated-column-spec	360
generation-clause	366
GRANT	367
INSERT	374
PIN TABLE	377
RENAME COLUMN	380
RENAME INDEX	382
RENAME TABLE	383
REVOKE	384
SELECT	393
SET ROLE	396
SET SCHEMA	398
TRUNCATE TABLE	400
UNPIN TABLE	401
UPDATE TABLE	403

Clauses

Introduction	405
CONSTRAINT	407
EXCEPT	415
FROM	418
GROUP BY	420
HAVING	422
LIMIT n	424
ORDER BY	427

OVER	430
RESULT OFFSET and FETCH FIRST	434
TOP n	436
UNION	440
USING	443
WHERE	445
WITH	447

Expressions

Introduction	449
About Expressions	450
Boolean Expressions	454
CASE Expression	459
Dynamic Parameters	460
Expression Precedence	463
NEXT VALUE FOR Expression	464
SELECT Expression	466
TABLE Expression	471
VALUES Expression	473

Join Operations

Introduction	477
About Join Operations	478
CROSS JOIN	480
INNER JOIN	482
LEFT OUTER JOIN	486
NATURAL JOIN	488
RIGHT OUTER JOIN	490

Queries

Introduction	492
Query	493
Scalar Subquery	495
Table Subquery	496

SQL Built-in Functions

Introduction	498
ABS	505
ACOS	506
ADD_MONTHS	508
ASIN	510

ATAN	512
ATAN2	514
AVG	516
BIGINT	520
CAST	522
CEIL	527
CHAR	529
COALESCE	532
Concatenation Operator	534
COS	536
COSH	538
COT	540
COUNT	542
CURRENT SCHEMA	545
CURRENT_DATE	546
CURRENT_ROLE	548
CURRENT_TIME	549
CURRENT_TIMESTAMP	550
CURRENT_USER	551
DATE	552
DAY	555
DEGREES	558
DENSE_RANK	560
DOUBLE	566
EXP	568
EXTRACT	569
FIRST_VALUE	574
FLOOR	576
HOUR	577
INITCAP	578
INSTR	580
INTEGER	582
LAG	584
LAST_DAY	587
LAST_VALUE	590
LCASE	592
LEAD	594
LENGTH	598
LN (or LOG)	601

LOCATE	602
LOG10.....	604
LTRIM.....	605
MAX	607
MIN.....	610
MINUTE	613
MOD.....	614
MONTH.....	618
MONTHNAME.....	620
MONTH_BETWEEN	623
NEXT_DAY	625
NOW	628
NULLIF.....	630
NVL	634
PI.....	636
QUARTER.....	637
RADIANS.....	640
RAND	642
RANDOM	643
RANK	644
REGEXP_LIKE	650
REPLACE	652
ROWID.....	654
ROW_NUMBER.....	656
RTRIM.....	658
SECOND.....	660
SESSION_USER	661
SIGN	662
SIN	663
SINH.....	665
SMALLINT.....	667
SQRT	669
STDDEV_POP	673
STDDEV_SAMP	675
SUBSTR.....	677
SUM	679
TAN	682
TANH.....	684
TIME	686

TIMESTAMP	688
TIMESTAMPADD	691
TIMESTAMPDIFF	693
TO_CHAR	695
TO_DATE	697
TRIM	705
TRUNCATE	708
UCASE or UPPER	714
USER	716
VARCHAR	717
WEEK	719
YEAR	721

System Procedures

Introduction	723
BACKUP_DATABASE	730
BULK_IMPORT_HFILE	733
CANCEL_BACKUP	739
CANCEL_DAILY_BACKUP	740
COLLECT_SCHEMA_STATISTICS	742
COMPACT_REGION	745
COMPUTE_SPLIT_KEY	748
CREATE_USER	753
DELETE_BACKUP	756
DELETE_OLD_BACKUPS	758
DELETE_REGION	760
DELETE_SNAPSHOT	764
DISABLE_COLUMN_STATISTICS	765
DROP_SCHEMA_STATISTICS	767
DROP_USER	769
EMPTY_GLOBAL_STATEMENT_CACHE	771
EMPTY_STATEMENT_CACHE	773
ENABLE_COLUMN_STATISTICS	775
ENABLE_ENTERPRISE	777
GET_ACTIVE_SERVERS	778
GET_ALL_PROPERTIES	779
GET_CURRENT_TRANSACTION	783
GET_GLOBAL_DATABASE_PROPERTY Function	784
GET_ENCODED_REGION_NAME	786
GET_LOGGERS	789

GET_LOGGER_LEVEL	795
GET_REGIONS	797
GET_REGIONS_SERVER_STATS_INFO.....	800
GET_REQUESTS	801
GET_RUNNING_OPERATIONS.....	802
GET_SCHEMA_INFO.....	804
GET_SESSION_INFO	806
GET_START_KEY	808
GET_VERSION_INFO.....	810
GET_WRITE_INTAKE_INFO.....	811
IMPORT_DATA	813
INSTALL_JAR	820
INVALIDATE_STORED_STATEMENTS	822
KILL_OPERATION.....	823
MAJOR_COMPACT_REGION.....	824
MERGE_DATA_FROM_FILE.....	827
MERGE_REGIONS	834
MODIFY_PASSWORD	836
PEEK_AT_SEQUENCE Function	838
PERFORM_MAJOR_COMPACTION_ON_SCHEMA.....	840
PERFORM_MAJOR_COMPACTION_ON_TABLE.....	841
REFRESH_EXTERNAL_TABLE.....	842
REMOVE_JAR	843
REPLACE_JAR.....	845
RESET_PASSWORD	847
RESTORE_DATABASE	849
RESTORE_SNAPSHOT	852
SCHEDULE_DAILY_BACKUP.....	853
SET_GLOBAL_DATABASE_PROPERTY	855
SET_LOGGER_LEVEL	857
SET_PURGE_DELETED_ROWS.....	859
SNAPSHOT_SCHEMA.....	860
SNAPSHOT_TABLE	861
SPLIT_TABLE_OR_INDEX.....	862
SPLIT_TABLE_OR_INDEX_AT_POINTS.....	867
UPDATE_ALL_SYSTEM_PROCEDURES	869
UPDATE_METADATA_STORED_STATEMENTS	871
UPDATE_SCHEMA_OWNER.....	872
UPDATE_SYSTEM_PROCEDURE	873

UPSERT_DATA_FROM_FILE	875
VACUUM	882

System Tables

Introduction	883
SYSALIASES system table	886
SYSBACKUP system table	887
SYSBACKUPITEMS system table	888
SYSBACKUPJOBS system table	889
SYSCHECKS system table	890
SYSCOLPERMS system table	891
SYSCOLUMNS system table	893
SYSCOLUMNSTATISTICS system table	895
SYSCONGLOMERATES system table	897
SYSCONSTRAINTS system table	899
SYSDEPENDS system table	900
SYSFILES system table	901
SYSFOREIGNKEYS system table	902
SYSKEYS system table	903
SYSPERMS system table	904
SYSROLES system table	905
SYSROUTINEPERMS system table	907
SYSSCHEMAS system table	908
SYSSEQUENCES system table	909
SYSSNAPSHOTS system table	911
SYSSTATEMENTS system table	912
SYSTABLEPERMS system table	913
SYSTABLES system table	915
SYSTABLESTATISTICS system table	916
SYSTRIGGERS system table	918
SYSUSERS system table	920
SYSVIEWS system table	921

Error Codes

Introduction	922
Class 01 - Warning	925
Class 07 - Dynamic SQL Error	927
Class 08 - Connection Exception	928
Class 0A - Feature not supported	931
Class 0P - Invalid role specification	932

Class 21 - Cardinality Violations	933
Class 22 - Data Exception	934
Class 23 - Constraint Violation.....	937
Class 24 - Invalid Cursor State	938
Class 25 - Invalid Transaction State	939
Class 28 - Invalid Authorization Specification	940
Class 2D - Invalid Transaction Termination.....	941
Class 38 - External Function Exception	942
Class 39 - External Routine Invocation Exception	943
Class 3B - Invalid SAVEPOINT	944
Class 40 - Transaction Rollback	945
Class 42 - Syntax Error or Access Rule Violation	946
Class 57 - DRDA Network Protocol Execution Failure	963
Class 58 - DRDA Network Protocol Protocol Error	964
Class XBCA - CacheService.....	966
Class XBCM - ClassManager	967
Class XBCX - Cryptography	968
Class XBM - Monitor	970
Class XCL - Execution exceptions	972
Class XCW - Upgrade unsupported.....	975
Class XCX - Internal Utility Errors	976
Class XCY - Splice Property Exceptions	977
Class XCZ - com.splicemachine.db.database.UserUtility	978
Class XD00 - Dependency Manager	979
Class XIE - Import/Export Exceptions	980
Class XJ - Connectivity Errors	982
Class XK - Security Exceptions	988
Class XN - Network Client Exceptions.....	989
Class XRE - Replication Exceptions	990
Class XSAI - Store access.protocol.interface	992
Class XSAM - Store AccessManager	993
Class XSAS - Store Sort	994
Class XSAX - Store access.protocol.XA statement	995
Class XSCB - Store BTree	996
Class XSCG0 - Conglomerate	997
Class XSCH - Heap	998
Class XSDA - RawStore Data.Generic statement	999
Class XSDB - RawStore Data.Generic transaction	1001
Class XSDF - RawStore Data.Filesystem statement.....	1002

Class XSDG - RawStore Data.Filesystem database	1003
Class XSLA - RawStore Log.Generic database exceptions	1004
Class XSLB - RawStore Log.Generic statement exceptions	1006
Class XSRS - RawStore protocol.Interface statement.....	1007
Class XSTA2 - XACT_TRANSACTION_ACTIVE	1008
Class XSTB - RawStore Transactions.Basic system	1009
Class XXXXX - No SQLSTATE	1010
Class X0 - Execution exceptions	1011

Splice Machine Developer's Guide

Developer's Guide

Introduction	1015
--------------------	------

Fundamentals

Introduction	1016
Running Transactions	1018
Working with Date and Time Values	1027
Using Database Triggers	1033
Using Foreign Keys.....	1037
Using Window Functions	1039
Using Temporary Tables	1050
Using Spark Libraries.....	1053
Using the Virtual Table Interface	1061
Using External Tables	1071
Using HCatalog	1075
Connecting Through HAProxy	1081
Using the MapReduce API.....	1086
Working with HBase	1090

Functions and Stored Procedures

Introduction	1093
Writing Functions and Stored Procedures	1097
Storing/Updating Functions and Procs	1102
Examples	1105

Tuning and Debugging

Introduction	1110
Optimizing Queries.....	1111
Using Statistics.....	1118

Using Explain Plan	1121
Explain Plan Examples	1124
Logging	1133
Debugging	1136
Using Snapshots	1138

Splice Machine Command Line Reference

Command Line Reference

Introduction	1139
Command Line Syntax	1142

Commands Available via All Connections

Analyze command	1150
Autocommit command	1153
Commit command	1154
Execute command	1155
Explain Plan command	1157
Export command	1158
Prepare command	1160
Release Savepoint command	1161
Remove Command	1164
Rollback command	1165
Rollback to Savepoint command	1166
Savepoint command	1168

Commands Only Available via sqlshell.sh

Connect command	1170
Describe command	1172
Disconnect command	1174
ElapsedTime command	1175
Exit command	1176
Help command	1177
MaximumDisplayWidth command	1179
Run command	1180
Set Connection command	1181
Show Connections command	1182
Show Functions command	1183
Show Indexes command	1184

Show PrimaryKeys command	1186
Show Procedures command	1187
Show Roles command	1191
Show Schemas command	1192
Show Synonyms command	1193
Show Tables command	1194
Show Views command	1196

Splice Machine Database Console

Database Console

Introduction	1197
Features	1200
Managing Queries	1203

Splice Machine Database-as-Service Product

Database-as-Service Product

Welcome!	1216
About our DBaaS	1217
Service Overview	1219
User Interface Overview	1221
Release Notes	1222

Cloud Manager Guide

Introduction	1223
Registering with Cloud Manager	1225
Logging into the Cloud Manager	1227
Creating Your Cluster	1228
Exploring Your Dashboard	1237
Managing a Cluster	1239
Managing Your Account	1244
Reviewing Event Notifications	1249

Using Zeppelin

Introduction	1251
Getting Started with Zeppelin	1252
Zeppelin Usage Notes	1258
A Simple Tutorial	1260

Splice Machine On-Premise Database Product

On-Premise Database Product

Welcome!	1265
Product Requirements	1266
Product Editions	1269
Release Notes	1270

Installing Splice Machine

Introduction	1272
Install Splice Machine	1278
Splice Machine AWS Sandbox	1280

On Premise Administration

Introduction	1292
Starting Your Database	1293
Backing Up	1296
Cleaning Your Database	1307
Shutting Down Your Database	1312
Derby Property Access	1314
Enabling Enterprise Edition	1317

Best Practices & TroubleShooting

Troubleshooting	1319
Configuration Option Info	1323

Splice Machine Release Notes

Splice Machine Database

Introduction	1325
New Features and Changes	1326
Improvements	1334
Issues Fixed	1340
Limitations and Workarounds	1355

Database-as-Service Product

Product Release Notes	1222
-----------------------	------

On Premise Database Product

Product Release Notes	1270
-----------------------	------

Splice Machine General Information

General Information

Introduction	1361
Splice Machine Editions	1362
Spark Overview.....	1364
Using our Documentation	1367
Documentation Examples Db	1370
Trademarks Information	1377
Glossary.....	1379

Splice Machine Tutorials

This guide includes tutorials to help you quickly become proficient in various aspects of using Splice Machine:

- » [Using the Splice Machine Command Line Interface](#)
- » [Configuring Splice Machine Security Settings](#)
- » [Importing Data](#)
- » [Accessing Data in the Cloud](#)
- » [Streaming Data](#)
- » [Connecting to Splice Machine with JDBC](#)
- » [Connecting to Splice Machine with ODBC](#)
- » [Connecting Splice Machine with Business Intelligence Tools](#)
- » [Analytics and Machine Learning](#)

Using the Splice Machine Command Line Interface

This section shows you how to use the `splice>` command line interface, in these topics:

- » Using the `splice>` command line interface
- » Scripting commands
- » Using r1Wrap

Configuring Splice Machine Security Settings

This section shows you how to configure security for use with Splice Machine, in these topics:

- » [Splice Machine Authentication and Authorization](#)
- » Configuring SSL/TLS Authentication
- » Using HAProxy on a Kerberos-Enabled Cluster

Importing Data

Our *Importing Data* tutorial walks you through importing data into Splice Machine, in these topics:

- » Overview of Importing Data
- » Import Parameters
- » Handling Import Input Data

- » Import Error Handling
- » Examples of Importing Data
- » Examples of Using Bulk HFile Import
- » Example: Importing TPCH Benchmark Data

Accessing Data in the Cloud

This section shows you how to access data that you have stored in the Cloud from Splice Machine, in these topics:

- » Configuring S3 Buckets for Splice Machine Access
- » Uploading Data to an S3 Bucket

Connecting to Splice Machine with JDBC

This section introduces our JDBC driver and shows you how to connect to Splice Machine via JDBC with various programming languages, including:

- » Connecting with Java and JDBC
- » Connecting with JRuby and JDBC
- » Connecting with Jython and JDBC
- » Connecting with Scala and JDBC
- » Connecting with AngularJS/NodeJS and JDBC

Connecting to Splice Machine with ODBC

This section introduces our ODBC driver and shows you how to connect to Splice Machine via ODBC with various programming languages, including:

- » Installing our ODBC Driver
- » Connecting with Python and ODBC
- » Connecting with C and ODBC

Streaming Data

This section contains topics that show you how to stream data into and out of Splice Machine:

- » Streaming MQTT Data
- » Integrating Apache Storm with Splice Machine

- » Configuring a Kafka Feed
- » Creating a Kafka Producer

Connecting Splice Machine with Business Intelligence Tools

This section shows you how to connect specific Business Intelligence tools to your Splice Machine database, including:

- » Connecting Cognos to Splice Machine
- » Connecting DBeaver to Splice Machine
- » Connecting DBVisualizer to Splice Machine
- » Connecting SQuirreL to Splice Machine
- » Connecting Tableau to Splice Machine

Analytics and Machine Learning

This section provides examples of using Splice Machine with various analytical and machine learning tools.

- » Connecting with Apache Zeppelin

Splice Machine Command Line Tutorials

This section shows you how to use the `splice>` command line interface, in these topics:

- » Using the `splice>` command line interface
- » Scripting commands
- » Using `rlWrap`

Getting Started With the `splice>` Command Line Interface

This is an On-Premise-Only topic! [Learn about our products](#)

The `splice>` command line interpreter is an easy way to interact with your Splice Machine database. This topic introduces `splice>` and some of the more common commands you'll use.



The command line interpreter, as documented here, is not available in our Cloud-Managed Database-as-Service product.

You can complete this tutorial by [watching a short video](#) or by [following the written version](#).

Watch the Video

The following video shows you how to launch and start using the `splice>` command line interpreter to connect to and interact with your database.

Follow the Written Version

This topic walks you through getting started with the `splice>` command line interpreter, in these sections:

- » [Starting `splice>`](#)
- » [Basic Syntax Rules](#)
- » [Connecting to a Database](#)
- » [Displaying Database Objects](#)
- » [Basic DDL and DML Statements](#)

NOTE: Although we focus here on executing command lines with the `splice>`, you can also use the command line interface to directly execute any SQL statement, including the DDL and DML statements that we introduce in the [last section](#) of this topic.

Starting `splice>`

To launch the `splice>` command line interpreter, follow these steps:

1. Open a terminal window

2. Navigate to your splicemachine directory

```
cd ~/splicemachine      #Use the correct path for your Splice Machine installation
```

3. Start splice>

```
./bin/sqlshell.sh
```

The full path to this script on Splice Machine standalone installations is `./splicemachine/bin/sqlshell.sh`.

4. The command line interpreter starts:

```
Running Splice Machine SQL ShellFor help: "Splice> help;"SPLICE** = current connectionsplice>
```

`SPLICE` is the name of the default connection, which becomes the current connection when you start the interpreter.

Restarting splice>

If you are running the standalone version of Splice Machine and your computer goes to sleep, any live database connections are lost. You'll need to restart Splice Machine by following these steps:

Step	Command
Exit splice>	splice> quit; (exit;)
Stop Splice Machine processes	\$./bin/stop-splice.sh
Restart Splice Machine processes	\$./bin/start-splice.sh
Restart splice>	\$./bin/sqlshell.sh

Basic Syntax Rules

When using the command line (the `splice>` prompt), you must end each SQL statement with a semicolon (`;`). For example:

```
splice> select * from myTable;
```

You can extend SQL statements across multiple lines, as long as you end the last line with a semicolon. Note that the `splice>` command line interface prompts you with a fresh `>` at the beginning of each line. For example:

```
splice> select * from myTable> where i > 1;
```

In most cases, the commands you enter are not case sensitive; you can Certain identifiers and keywords are case sensitive: this means that these commands are all equivalent:

```
splice> show connections;
splice> SHOW CONNECTIONS;
splice> Show Connections;
```

The *Command Line Syntax* topic contains a complete syntax reference for `splice>`.

Connecting to a Database

When you start `splice>`, you are automatically connected to your default database. You can connect to other databases with the `connect` command:

```
connect 'jdbc:splice://srv55:1527/splicedb;user=splice;password=admin' AS DEMO;
```

Anatomy of a Connection String

Here's how to breakdown the connection strings we use to connect to a database:

Examples	Component	Comments
<code>jdbc:splice:</code>	Connection driver name	
<code>srv55:1527</code> <code>localhost:1527</code>	Server Name:Port	<code>splice></code> listens on port 1527
<code>splicedb</code>	Database name	The name of the database you're connecting to on the server.
<code>user=splice;password=admin</code>	Connection parameters	Any required connection parameters, such as userId and password.
<code>AS DEMO</code>	Optional connection identifier	The name that you want associated with this connection. If you don't supply a name, Splice Machine assigns one for you; for example: CONNECTION1.

Displaying Database Objects

We'll first explore the `show` command, which is available to view numerous object types in your database, including: [connections](#), [schemas](#), [tables](#), [indexes](#), [views](#), [procedures](#), and others.

Displaying and Changing Connections

You can connect to multiple database in Splice Machine; one connection is designated as the current database; this is the database with which you're currently working.

To view your current connections, use the `show connections` command:

```
splice> show connections;
DEMO      - jdbc:splice://srv55:1527/splicedb
SPLICE*   - jdbc:splice://localhost:1527/splicedb
* = current connection
```

You can use the `set connection` command to modify the current connection:

```
splice> SET CONNECTION DEMO;
splice> show connections;
DEMO*      - jdbc:splice://srv55:1527/splicedb
SPLICE     - jdbc:splice://localhost:1527/spicedb
* = current connection
```

You can use the `disconnect` command to close a connection:

```
splice> Disconnect DEMO;
splice> show Connections;
SPLICE    - jdbc:splice://localhost:1527/spicedb
No current connection
```

Notice that there's now no current connection because we've disconnected the connection named `DEMO`, which had been the current connection. We can easily resolve this by connecting to a named connection:

```
splice> connect splice;
splice> show connections;
SPLICE*   - jdbc:splice://localhost:1527/spicedb
* = current connection
```

Finally, to disconnect from all connections:

```
splice> disconnect all;
splice> show connections;
No connections available
```

Displaying Schemas

Use the `show schemas` command to display the schemas that are defined in your currently connected database:

```
splice> show schemas;
TABLE_SCHEM
-----
NULLID
SPLICE
SQLJ
SYS
SYSCAT
SYSCS_DIAG
SYSCS_UTIL
SYSFUN
SYSIBM
SYSPROC
SYSSTAT
11 rows selected
```

The current schema is used as the default value when you issue commands that optionally take a schema name as a parameter. For example, you can optionally specify a schema name in the show tables command; if you don't include a schema name, Splice Machine assumes the current schema name.

To display the current schema name, use the built-in `current schema` function:

```
splice> values(current schema);
1
-----
SPLICE
1 row selected
```

To change which schema is current, use the SQL `set schema` statement:

```
splice> set schema SYS;
0 rows inserted/updated/deleted
splice> values(current schema);
1
-----
SYS
1 row selected
```

Displaying Tables

The `show tables` command displays a list of all tables in all of the schemas in your database:

```
splice> SHOW TABLES;
TABLE_SCHEM | TABLE_NAME | CONGLOM_ID | REMARKS
-----
SYS | SYSALIASES | 256 |
SYS | SYSBACKUP | 992 |
SYS | SYSBACKUPFILESET | 1008 |
SYS | SYSBACKUPITEMS | 1104 |
SYS | SYSBACKUPJOBS | 1216 |
SYS | SYSCHECKS | 336 |
SYS | SYSCOLPERMS | 608 |
SYS | SYSCOLUMNS | 80 |
SYS | SYSCOLUMNSTATS | 1264 |
SYS | SYSCONGLOMERATES | 48 |
SYS | SYSCONSTRAINTS | 304 |
SYS | SYSDEPENDS | 368 |
SYS | SYSFILES | 288 |
SYS | SYSFOREIGNKEYS | 272 |
SYS | SYSKEYS | 240 |
SYS | SYSPERMS | 912 |
SYS | SYSPHYSICALSTATS | 1280 |
SYS | SYSPRIMARYKEYS | 320 |
SYS | SYSROLES | 816 |
SYS | SYSROUTINEPERMS | 656 |
SYS | SYSSCHEMAPERMS | 1328 |
SYS | SYSSCHEMAS | 32 |
SYS | SYSEQUENCES | 864 |
SYS | SYSSTATEMENTS | 384 |
SYS | SYSTABLEPERMS | 592 |
SYS | SYSTABLES | 64 |
SYS | SYSTABLESTATS | 1296 |
SYS | SYSTRIGGERS | 576 |
SYS | SYSUSERS | 928 |
SYS | SYSVIEWS | 352 |
SYSIBM | SYSDUMMY1 | 1312 |
SPLICE | CUSTOMERS | 1568 |
SPLICE | T_DETAIL | 1552 |
SPLICE | T_HEADER | 1536 |

34 rows selected
```

To display the tables in a specific schema (named SPLICE):

```
splice> show tables in SPLICE;
TABLE_SCHEM | TABLE_NAME | CONGLOM_ID | REMARKS
-----
SPLICE | CUSTOMERS | 1568 |
SPLICE | T_DETAIL | 1552 |
SPLICE | T_HEADER | 1536 |

3 rows selected
```

To examine the structure of a specific table, use the `DESCRIBE` command:

```
splice> describe T_DETAIL;
COLUMN_NAME          | TYPE_NAME | DEC | NUM | COLUM | COLUMN_DEF | CHAR_OCTE | IS_NUL
L
-----
TRANSACTION_HEADER_KEY | BIGINT    | 0   | 10  | 19   | NULL      | NULL      | NO
TRANSACTION_DETAIL_KEY | BIGINT    | 0   | 10  | 19   | NULL      | NULL      | NO
CUSTOMER_MASTER_ID    | BIGINT    | 0   | 10  | 19   | NULL      | NULL      | YES
TRANSACTION_DT        | DATE      | 0   | 10  | 10   | NULL      | NULL      | NO
ORIGINAL_SKU_CATEGORY_ID | INTEGER  | 0   | 10  | 10   | NULL      | NULL      | YES
5 rows selected
```

Displaying Indexes

You can display all of the indexes in a schema:

```
splice> show indexes in SPLICE;
TABLE_NAME      | INDEX_NAME      | COLUMN_NAME          | ORDINAL& | NON_UNIQUE | TYPE  | ASC& | CO
NGLOM_NO
-----
T_DETAIL         | TDIDX1          | ORIGINAL_SKU_CATEG& | 1         | true       | BTREE | A     | 15
85
T_DETAIL         | TDIDX1          | TRANSACTION_DT      | 2         | true       | BTREE | A     | 15
85
T_DETAIL         | TDIDX1          | CUSTOMER_MASTER_ID  | 3         | true       | BTREE | A     | 15
85
T_HEADER         | THIDX2          | CUSTOMER_MASTER_ID  | 1         | true       | BTREE | A     | 16
01
T_HEADER         | THIDX2          | TRANSACTION_DT      | 2         | true       | BTREE | A     | 16
01
5 rows selected
```

Or you can display the indexes defined for a specific table:

```
splice> show indexes FROM T_DETAIL;
TABLE_NAME | INDEX_NAME | COLUMN_NAME | ORDINAL& | NON_UNIQUE | TYPE | ASC& | CO
NGLOM_NO
-----
T_DETAIL    | TDIDX1      | ORIGINAL_SKU_CATEG& | 1          | true       | BTREE | A   | 15
85
T_DETAIL    | TDIDX1      | TRANSACTION_DT     | 2          | true       | BTREE | A   | 15
85
T_DETAIL    | TDIDX1      | CUSTOMER_MASTER_ID | 3          | true       | BTREE | A   | 15
85
3 rows selected
```



Note that we use IN to display the indexes in a schema, and FROM to display the indexes in a table.

Displaying Views

Similarly to indexes, you can use the `show views` command to display all of the indexes in your database or in a schema:

```
splice> show views;
TABLE_SCHEM | TABLE_NAME | CONGLOM_ID | REMARKS
-----
SYS         | SYSCOLUMNSTATISTICS | NULL      |
SYS         | SYSTABLESTATISTICS   | NULL      |

2 rows selected
splice> show views in sys;TABLE_SCHEM | TABLE_NAME | CONGLOM_ID | REMARKS
-----
SYS         | SYSCOLUMNSTATISTICS | NULL      |
SYS         | SYSTABLESTATISTICS   | NULL      |

2 rows selected
```

Displaying Stored Procedures and Functions

You can create *user-defined database functions* that can be evaluated in SQL statements; these functions can be invoked where most other built-in functions are allowed, including within SQL expressions and SELECT statement. Functions must be deterministic, and cannot be used to make changes to the database. You can use the `show functions` command to display which functions are defined in your database or schema:

```
splice> show functions in splice;
FUNCTION_SCHEM | FUNCTION_NAME | REMARKS
-----
SPICE          | TO_DEGREES      | java.lang.Math.toDegrees
1 row selected
```

You can also group a set of SQL commands together with variable and logic into a *stored procedure*, which is a subroutine that is stored in your database's data dictionary. Unlike user-defined functions, a stored procedure is not an expression and can only be invoked using the `CALL` statement. Stored procedures allow you to modify the database and return Result Sets or nothing at all. You can use the `show procedures` command to display which functions are defined in your database or schema:

```
splice> show procedures in SQLJ;
PROCEDURE_SCHEM | PROCEDURE_NAME | REMARKS
-----
SQLJ             | INSTALL_JAR      | com.splicemachine.db.catalog.SystemProcedure
s.INSTALL_JAR   |
SQLJ             | REMOVE_JAR       | com.splicemachine.db.catalog.SystemProcedure
s.REMOVE_JAR    |
SQLJ             | REPLACE_JAR      | com.splicemachine.db.catalog.SystemProcedure
s.REPLACE_JAR   |

3 rows selected
```

Basic DDL and DML Statements

This section introduces the basics of running SQL Data Definition Language (*DDL*) and Data Manipulation Language (*DML*) statements from `splice>`.

- » [Getting Started With the `splice>` Command Line Interface](#)
- » [Getting Started With the `splice>` Command Line Interface](#)
- » [Inserting Data](#)
- » [Selecting and Displaying Data](#)

See the *DML Statements* sections in our *SQL Reference Manual* for more information.

CREATE Statements

SQL uses `CREATE` statements to create objects such as tables. For example:

```

splice> CREATE schema MySchema1;
0 rows inserted/updated/deleted
splice> create Schema mySchema2;
0 rows inserted/updated/deleted
splice> show schemas;
TABLE_SCHEM
-----
MYSCHHEMA1MYSCHEMA2NULLID
SPLICE
SQLJ
SYS
SYSCAT
SYSICS_DIAG
SYSICS_UTIL
SYSFUN
SYSIBM
SYSPROC
SYSSTAT
13 rows selected
splice> SET SCHEMA MySchema1;
0 rows inserted/updated/deleted
splice> CREATE TABLE myTable ( myNum int, myName VARCHAR(64) );
0 rows inserted/updated/deleted
splice> CREATE TABLE Players(
    ID          SMALLINT NOT NULL PRIMARY KEY,
    Team        VARCHAR(64) NOT NULL,
    Name        VARCHAR(64) NOT NULL,
    Position    CHAR(2),
    DisplayName VARCHAR(24),
    BirthDate   DATE
);
0 rows inserted/updated/deleted
splice> SHOW TABLES IN MySchema1;
TABLE_SCHEM | TABLE_NAME           | CONGLOM_ID | REMARKS
-----
MYSCHHEMA1  | MYTABLE                | 1616      |
MYSCHHEMA1  | PLAYERS                 | 1632      |
2 rows selected

splice> describe Players;
COLUMN_NAME | TYPE_NAME | DEC& | NUM& | COLUM& | COLUMN_DEF | CHAR_OCTE& | IS_NULL&
-----
ID          | SMALLINT  | 0    | 10   | 5     | NULL      | NULL      | NO
TEAM        | VARCHAR   | NULL | NULL  | 64    | NULL      | 128       | NO
NAME        | VARCHAR   | NULL | NULL  | 64    | NULL      | 128       | NO
POSITION    | CHAR      | NULL | NULL  | 2     | NULL      | 4         | YES
DISPLAYNAME | VARCHAR   | NULL | NULL  | 24    | NULL      | 48        | YES
BIRTHDATE   | DATE      | 0    | 10   | 10   | NULL      | NULL      | YES
6 rows selected

```

See the *CREATE Statements* section in our *SQL Reference Manual* for more information.

DROP Statements

SQL uses DROP statements to delete objects such as tables. For example:

```
splice> DROP schema MySchema2 restrict;0 rows inserted/updated/deleted
```

You **must** include the keyword `restrict` when dropping a schema; this enforces the rule that the schema cannot be deleted from the database if there are any objects defined in the schema.

```
splice> show schemas;
TABLE_SCHEM
-----
MYSCHHEMA1
MYSCHHEMA2
NULLID
SPLICE
SQLJ
SYS
SYSCAT
SYSCS_DIAG
SYSCS_UTIL
SYSFUN
SYSIBM
SYSPROC
SYSSTAT
12 rows selected

splice> DROP TABLE myTable;
0 rows inserted/updated/deleted
splice> SHOW TABLES IN MySchema1;
TABLE_SCHEM | TABLE_NAME | CONGLOM_ID | REMARKS
-----
MYSCHHEMA1 | PLAYERS | 1632 | 1 row selected
```

See the *DROP Statements* section in our *SQL Reference Manual* for more information.

Inserting Data

Once you've created a table, you can use `INSERT` statements to insert records into that table; for example:

```
splice> INSERT INTO Players
    VALUES( 99, 'Giants', 'Joe Bojangles', 'C', 'Little Joey', '07/11/1991');
1 row inserted/updated/deleted

splice> INSERT INTO Players
    VALUES (99, 'Giants', 'Joe Bojangles', 'C', 'Little Joey', '07/11/1991'),
            (73, 'Giants', 'Lester Johns', 'P', 'Big John', '06/09/1984'),
            (27, 'Cards', 'Earl Hastings', 'OF', 'Speedy Earl', '04/22/1982');
3 rows inserted/updated/deleted
```

Selecting and Displaying Data

Now that you have a bit of data in your table, you can use **SELECT** statements to select specific records or portions of records. This section contains several simple examples of selecting data from the `Players` table we created in the previous section.

You can select a single column from all of the records in a table; for example:

```
splice> select NAME from Players;
NAME
-----
Earl Hastings
Lester Johns
Joe Bojangles
```

3 rows selected

You can select all columns from all of the records in a table; for example:

```
splice> select * from Players;
ID |TEAM      |NAME          |POS&|DISPLAYNAME      |BIRTHDATE
-----
27 |Cards     |Earl Hastings |OF   |Speedy Earl       |1982-04-22
73 |Giants    |Lester Johns  |P    |Big John          |1984-06-09
99 |Giants    |Joe Bojangles |C    |Little Joey        |1991-07-11
```

3 rows selected

You can also qualify which records to select with a **WHERE** clause; for example:

```
splice> select * from Players WHERE Team='Cards';
ID |TEAM      |NAME          |POS&|DISPLAYNAME      |BIRTHDATE
-----
27 |Cards     |Earl Hastings |OF   |Speedy Earl       |1982-04-22
1 row selected
```

You can easily count the records in a table by using the **SELECT** statement; for example:

```
splice> select count(*) from Players;
-----
31 rows selected
```

See Also

- » To learn how to script `splice>` commands, please see the Scripting Splice Machine Commands tutorial.
- » For more information about the `splice>` command line interpreter, see the Command Line Reference Manual, which includes information about and examples of all supported commands.
- » This documentation includes a number of other tutorials to help you become proficient with Splice Machine.

Scripting the splice> Command Line Interface

This is an On-Premise-Only topic! [Learn about our products](#)

You can use two simple and different methods to script the `splice>` command line interpreter; both of described here:

- » [Running a File of splice> Commands](#)
- » [Running Splice Machine From a Shell Script](#)

Running a File of splice> Commands

You can create a simple text file of command lines and use the `splice> run` command to run the commands in that file. Follow these steps:

1. Create a file of SQL commands:

First, create a file that contains any SQL commands you want to run against your Splice Machine database.

For this example, we'll create a file named `mySQLScript.sql` that connects to a database, creates a table, inserts records into that table, and then displays the records in the table.

```
connect 'jdbc:splice://localhost:1527/splicedb;user=splice;password=admin';

create table players (
    ID SMALLINT NOT NULL PRIMARY KEY,
    Team VARCHAR(64) NOT NULL,
    Name VARCHAR(64) NOT NULL,
    Position CHAR(2),
    DisplayName VARCHAR(24),
    BirthDate DATE );

INSERT INTO Players
    VALUES (99, 'Giants', 'Joe Bojangles', 'C', 'Little Joey', '07/11/1991'),
           (73, 'Giants', 'Lester Johns', 'P', 'Big John', '06/09/1984'),
           (27, 'Cards', 'Earl Hastings', 'OF', 'Speedy Earl', '04/22/1982');

SELECT * FROM Players;
```

2. Start splice>

If you've not yet done so, start Splice Machine and the `splice>` command line interface. If you don't know how to do so, please see our [Introduction to the splice> Command Line Interface](#).

3. Run the SQL Script

Now, in `splice>`, run your script with the `run` command:

```
run 'mySQLScript.sql';
```

You'll notice that `splice>` displays exactly the same results as you would see if you typed each command line into the interface:

```
splice> connect 'jdbc:splice://localhost:1527/splicedb;user=splice;password=admin';
splice> create table players (
    ID SMALLINT NOT NULL PRIMARY KEY,
    Team VARCHAR(64) NOT NULL,
    Name VARCHAR(64) NOT NULL,
    Position CHAR(2),
    DisplayName VARCHAR(24),
    BirthDate DATE );
0 rows inserted/updated/deleted
splice> INSERT INTO Players
    VALUES (99, 'Giants', 'Joe Bojangles', 'C', 'Little Joey', '07/11/1991'),
           (73, 'Giants', 'Lester Johns', 'P', 'Big John', '06/09/1984'),
           (27, 'Cards', 'Earl Hastings', 'OF', 'Speedy Earl', '04/22/1982');
3 rows inserted/updated/deleted
splice> SELECT * FROM Players;
+----+-----+-----+-----+-----+
| ID | TEAM | NAME | POS & | DISPLAYNAME | BIRTHDATE |
+----+-----+-----+-----+-----+
| 27 | Cards | Earl Hastings | OF | Speedy Earl | 1982-04-22 |
| 73 | Giants | Lester Johns | P | Big John | 1984-06-09 |
| 99 | Giants | Joe Bojangles | C | Little Joey | 1991-07-11 |
+----+-----+-----+-----+-----+
3 rows selected
splice>
```

Running Splice Machine From a Shell Script

You can also use a shell script to start the `splice>` command line interpreter and run command lines with Unix heredoc (`<<`) input redirection. For example, we can easily rework the SQL script we used in the previous section into a shell script that starts `splice>`, runs several commands/statements, and then exits `splice>`.

1. Create a shell script

For this example, we'll create a file named `myShellScript.sql` that uses the same commands as we did in the previous example:

```
#!/bin/bash
echo "Running splice> commands from a shell script"./bin/sqlshell.sh << EOF
connect 'jdbc:splice://localhost:1527/splicedb;user=splice;password=admin';

create table players (
    ID SMALLINT NOT NULL PRIMARY KEY,
    Team VARCHAR(64) NOT NULL,
    Name VARCHAR(64) NOT NULL,
    Position CHAR(2),
    DisplayName VARCHAR(24),
    BirthDate DATE );

INSERT INTO Players
VALUES (99, 'Giants', 'Joe Bojangles', 'C', 'Little Joey', '07/11/1991'),
       (73, 'Giants', 'Lester Johns', 'P', 'Big John', '06/09/1984'),
       (27, 'Cards', 'Earl Hastings', 'OF', 'Speedy Earl', '04/22/1982);

SELECT * FROM Players;exit;EOF
```

If you're not familiar with this kind of input redirection: the `<<` specifies that an interactive program (`./bin/sqlshell.sh`) will receive its input from the lines in the file until it encounters EOF. The program responds exactly as it would had a user directly typed in those commands.

2. Make your script executable

Be sure to update permissions on your script file to allow it to run:

```
chmod +x myShellScript.sh
```

3. Run the script

In your terminal window, invoke the script:

```
./myShellScript.sh
```

You'll notice that `splice>` starts and runs exactly as it did in the SQL script example above, then exits.

Running Splice Machine Commands from a Shell Script...

```
===== rlwrap detected and enabled. Use up and down arrow keys to scroll through command line history. =====
```

```
Running Splice Machine SQL shell
For help: "splice> help;"splice> connect 'jdbc:splice://srv55:1527/splice
db;user=splice;password=admin';
splice> create table players (
    ID SMALLINT NOT NULL PRIMARY KEY,
    Team VARCHAR(64) NOT NULL,
    Name VARCHAR(64) NOT NULL,
    Position CHAR(2),
    DisplayName VARCHAR(24),
    BirthDate DATE );
0 rows inserted/updated/deleted
splice> INSERT INTO Players
    VALUES (99, 'Giants', 'Joe Bojangles', 'C', 'Little Joey', '07/11/199
1'),
        (73, 'Giants', 'Lester Johns', 'P', 'Big John', '06/09/1984'),
        (27, 'Cards', 'Earl Hastings', 'OF', 'Speedy Earl', '04/22/198
2');
3 rows inserted/updated/deleted
splice> SELECT * FROM Players;
ID      |TEAM          |NAME           | POS&|DISPLAYNAME | B
IRTHDATE
-----
-----
```

ID	TEAM	NAME	POS&	DISPLAYNAME	B
27	Cards	Earl Hastings	OF	Speedy Earl	1
982-04-22					
73	Giants	Lester Johns	P	Big John	1
984-06-09					
99	Giants	Joe Bojangles	C	Little Joey	1
991-07-11					

```
3 rows selected
```

Using nohub for Long-Running Scripts

If you want to run an unattended shell script that may take a long time, you can: use the Unix nohup utility, which allows you to start a script in the background and redirect its output. This means that you can start the script, log out, and view the output at a later time. For example:

```
nohup ./myShellScript.sh > ./myShellScript.out 2>&1 &
```

Once you've issued this command, you can log out, and subsequently view the output of your script in the myShellScript.out file.

rlWrap Commands Synopsis

This is an On-Premise-Only topic! [Learn about our products](#)

The *rlWrap* program is a *readline wrapper*, a small utility that uses the GNU *readline* library to allow the editing of keyboard input for any command; it also provides a history mechanism that is very handy for fixing or reusing commands. Splice Machine strongly recommends that you use *rlWrap* when interacting with your database via our command line interface, which is also known as the `splice>` prompt.

NOTE: You can customize many aspects of *rlWrap* and *readline*, including the keyboard bindings for the available commands. For more information, see the Unix man page for *readline*.

The following table summarizes some of the common keyboard options you can use with *rlWrap*; this table uses the default bindings that are in place when you install *rlWrap* on MacOS; keyboard bindings may be different in your environment.

Command	Description
CTRL-@	Set mark
CTRL-A	Move to the beginning of the line
CTRL-B	Move back one character
CTRL-D	Delete the highlighted character
CTRL-E	Move to the end of the line
CTRL-F	Move forward one character
CTRL-H	Backward delete character
CTRL-J	Accept (submit) the line
CTRL-L	Clear the screen
CTRL-M	Accept the line
CTRL-N	Move to the next line in history
CTRL-P	Move to the previous line in history
CTRL-R	Reverse search through your command line history
CTRL-S	Forward search through your command line history

Command	Description
CTRL-T	Transpose characters: switch the highlighted character with the one preceding it
CTRL-U	Discard from the cursor position to the beginning of the line
CTRL-]	Search for a character on the line
CTRL- _	Undo
ALT-<	Go to the beginning of the history
ALT->	Go to the end of the history
ALT-B	Backward word
ALT-C	Capitalize the current word
ALT-F	Forward word
ALT-L	Downcase word
ALT-R	Revert line
ALT-T	Transpose words
ALT-U	Uppercase word

Note that the `ALT` key is labeled as the `option` key on Macintosh keyboards.

NOTE: If you're using the `splice>` prompt in the Terminal.app on MacOS, the `ALT-` commands listed above only work if you select the `Use Option as Meta key` setting in the keyboard preferences for your terminal window

Configuring Splice Machine Security Settings

This section shows you how to configure security for Splice Machine, in these topics:

- » [Roles and Authorization](#)
- » Configuring Authentication
- » Configuring SSL/TLS
- » Using HAProxy with Kerberos

Splice Machine Authorization and Roles

This topic describes Splice Machine *user authorization*, which is how Splice Machine authorizes which operations can be performed by which users.

NOTE: The on-premise version of Splice Machine offers several different authentication mechanisms for your database, as described in the Configuring Splice Machine Authentication topic. Native authentication is the default mechanism.

With our built-in native authentication mechanism, the user that requests a connection must provide a valid name and password, which Splice Machine verifies against the repository of users defined for the system. After Splice Machine authenticates the user as valid, user authorization determines what operations the user can perform on the database to which the user is requesting a connection.

Managing Users

Splice manages users with standard system procedures:

- » You can create a user with the `SYSCS_UTIL.SYSCS_CREATE_USER` procedure:

```
splice> call syscs_util.syscs_create_user('username', 'password');
```

- » You can drop a user with the `SYSCS_UTIL.SYSCS_DROP_USER` procedure:

```
splice> call syscs_util.syscs_drop_user('username');
```

Managing Roles

When standard authorization mode is enabled, object owners can use roles to administer privileges. Roles are useful for administering privileges when a database has many users. Role-based authorization allows an administrator to grant privileges to anyone holding certain roles, which is less tedious and error-prone than administrating those privileges to a large set of users.

The Database Owner

The *database owner* is `splice`. Only the database owner can create, grant, revoke, and drop roles. However, object owners can grant and revoke privileges for those objects to and from roles, as well as to and from individual users and to `PUBLIC` (all users).

If authentication and SQL authorization are both enabled, only the database owner can perform these actions on the database:

- » start it up
- » shut it down
- » perform a full upgrade

If authentication is not enabled, and no user is supplied, the database owner defaults to SPLICE, which is also the name of the default schema.

The database owner log-in information for Splice Machine is configured when your database software is installed. If you're using our database as a service, there is no default userId or password; if you're using our on-premise database, the default userID is `splice`, and the default password is `admin`.

Creating and Using Roles

The database owner can use the `GRANT` statement to grant a role to one or more users, to `PUBLIC`, or to another role. Roles can be contained within other roles and can inherit privileges from roles that they contain.

Setting Roles

When a user first connects to Splice Machine, no role is set, and the `SET ROLE` statement to set the current role for that session. The role can be any role that has been granted to the session's current user or to `PUBLIC`.

To unset the current role, you can call `SET ROLE` with an argument of `NONE`. At any time during a session, there is always a current user, but there is a current role only if `SET ROLE` has been called with an argument other than `NONE`. If a current role is not set, the session has only the privileges granted to the user directly or to `PUBLIC`.

Roles in Stored Procedures and Functions

Within stored procedures and functions that contain SQL, the current role depends on whether the routine executes with invoker's rights or with definer's rights, as specified by the `EXTERNAL SECURITY` clause in the `CREATE PROCEDURE` statements. During execution, the current user and current role are kept on an authorization stack which is pushed during a stored routine call.

- » Within routines that execute with invoker's rights, the following applies: initially, inside a nested connection, the current role is set to that of the calling context. So is the current user. Such routines may set any role granted to the invoker or to `PUBLIC`.
- » Within routines that execute with definer's rights, the following applies: initially, inside a nested connection, the current role is `NULL`, and the current user is that of the definer. Such routines may set any role granted to the definer or to `PUBLIC`.

Upon return from the stored procedure or function, the authorization stack is popped, so the current role of the calling context is not affected by any setting of the role inside the called procedure or function. If the stored procedure opens more than one nested connection, these all share the same (stacked) current role (and user) state. Any dynamic result set passed out of a stored procedure sees the current role (or user) of the nested context.

Dropping Roles

Only the database owner can drop a role. To drop a role, use the `DROP ROLE` statement. Dropping a role effectively revokes all grants of this role to users and other roles.

Granting Privileges

Use the `GRANT` statement to grant privileges on schemas, tables and routines to a role or to a user.

Note that when you grant privileges to a role, you are implicitly granting those same privileges to all roles that contain that role.

Revoking Privileges

Use the [REVOKE](#) statement to revoke privileges on schemas, tables and routines.

When a privilege is revoked from a user:

- » That session can no longer keep the role, nor can it take on that role unless the role is also granted to PUBLIC.
- » If that role is the current role of an existing session, the current privileges of the session lose any extra privileges obtained through setting that role.

The default revoke behavior is CASCADE, which means that all persistent objects (constraints and views, views and triggers) that rely on a dropped role are dropped. Although there may be other ways of fulfilling that privilege at the time of the revoke, any dependent objects are still dropped. Any prepared statement that is potentially affected will be checked again on the next execute. A result set that depends on a role will remain open even if that role is revoked from a user.

See Also

- » [Configuring Splice Machine Authentication](#)
- » [CREATE FUNCTION](#)
- » [CREATE PROCEDURE](#)
- » [CREATE ROLE](#)
- » [CURRENT_ROLE](#)
- » [DROP ROLE](#)
- » [GRANT](#)
- » [REVOKE](#)
- » [SET ROLE](#)
- » [SYSCS_UTIL.SYSCS_CREATE_USER](#)
- » [SYSCS_UTIL.SYSCS_DROP_USER](#)

Configuring Splice Machine Authentication

This is an On-Premise-Only topic! [Learn about our products](#)

This topic describes the mechanisms you can use in Splice Machine to authenticate users and how to configure the mechanism you choose to use, in these sections:

- » [Supported Authentication Mechanisms](#)
- » [Configuring Authentication](#)

Supported Authentication Mechanisms

You can use one of the following authentication mechanisms, each of which is described below the table:

Authentication Mechanism	Description
None	Any user ID and password combination is allowed to connect to database.
Native	<p>User IDs in a database table are validated against the corresponding, encrypted password.</p> <p>This is the default authentication setting for Splice Machine installations.</p>
LDAP	<p>User IDs are validated against an existing LDAP service.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  ENTERPRISE ONLY: This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product. <p>You cannot use this feature with the Community editions of Splice Machine. For a list of the additional features available in the Enterprise edition, see our Splice Machine Editions page.</p> <p>To obtain a license for the Splice Machine <i>Enterprise Edition</i>, please Contact Splice Machine Sales today.</p> </div>

Configuring Authentication

You configure Splice Machine authentication by adding or updating properties in your HBase configuration file; this is typically done during installation of Splice Machine, but you can modify your settings whenever you want. This section contains the following subsections:

- » [Locating Your Configuration File](#)
- » [Disabling Authentication](#)
- » [Using Native Authentication](#)
- » [Using LDAP Authentication](#)

Locating Your Configuration File

The following table specifies the platform-specific location of the configuration you need to update when changing your Splice Machine authentication properties:

Platform	Configuration file to modify with your authentication properties
CDH	hbase-site.xml
HDP	Select the Custom HBase Configs option from the HBase configuration tab.
MapR	hbase-site.xml
Standalone version	splicemachine/lib/splice-site.xml

Configure your authentication settings by adding or modifying properties in the configuration file.

Disabling Authentication

If you want to disable authentication for your Splice Machine database, you can set the authentication property to NONE.

NOTE: Splice Machine **strongly encourages you to not use** an open database for production databases!

You can configure an open database that allows any user to authenticate against the database by setting your authentication properties as follows:

```
<property>
  <name>splice.authentication</name>
  <value>NONE</value>
</property>
```

Using Native Authentication

Native authentication is the default mechanism for Splice Machine; you don't need to modify your configuration if you wish to use it. Native authentication uses the `sys.sysusers` table in the `splice` schema for configuring user names and passwords.

The default native authentication property settings are:

```
<property>
  <name>splice.authentication</name>
  <value>NATIVE</value>
</property>
<property>
  <name>splice.authentication.native.algorithm</name>
  <value>SHA-512</value>
</property>
```

You can use MD5, SHA-256, or SHA-512 for the value of the `native.algorithm` property; SHA-512 is the default value.

Switching to Native Authentication

If you are switching your authentication from to Native authentication from another mechanism (including NONE), there's one additional step you need to take: you must re-initialize the credentials database (SYSUSERS table), by adding the following property setting to your configuration file:

```
<property>
  <name>splice.authentication.native.create.credentials.database</name>
  <value>true</value>
</property>
```

Using LDAP Authentication

LDAP authentication in Splice Machine uses an external LDAP server.



LDAP authentication is available only with a Splice Machine Enterprise license; you cannot use LDAP authentication with the Community version of Splice Machine.

To obtain a license for the Splice Machine Enterprise Edition, please [Contact Splice Machine Sales today](#).

To use LDAP with Splice Machine, you must:

- » [Contact us](#) to obtain a license key from Splice Machine.
- » Enable Enterprise features by adding your Splice Machine license key to your HBase configuration file as the value of the `splicemachine.enterprise.key` property, as shown below.
- » Make sure that a user with name `splice` has been created in the LDAP server.

- » Add the Splice Machine LDAP properties in your HBase configuration file, along with the license key property. Note that you may need to set `splice.authentication` properties in both service and client HBase configuration files:

LDAP Property Settings

These are the property settings you need to configure:

```
<property>
  <name>splicemachine.enterprise.key</name>
  <value><your-Splice-Machine-license-key></value>
</property>
<property>
  <name>splice.authentication</name>
  <value>LDAP</value>
</property>
<property>
  <name>splice.authentication.ldap.server</name>
  <value><ldap://servername-ldap.yourcompany.com:389></value>
</property>
<property>
  <name>splice.authentication.ldap.searchAuthDN</name>
  <value><cn=commonName,ou=Users,dc=yourcompany,dc=com></value>
</property>
<property>
  <name>splice.authentication.ldap.searchAuthPW</name>
  <value><yourpassword></value>
</property>
<property>
  <name>splice.authentication.ldap.searchBase</name>
  <value>ou=Users,dc=yourcompany,dc=com</value>
</property>
<property>
  <name>splice.authentication.ldap.searchFilter</name>
  <value>&lt;(&objectClass=*)(uid=%USERNAME%)&gt;</value>
</property>
```

Notes about the LDAP property values:

- Specify the location of your external LDAP server host in the `splice.authentication.ldap.server` property on port [389](#).
- The `ldap.searchAuthDN` property is the security principal:
 - This is used to create the initial LDAP context (aka its connection to a specific DN).
 - It must have the authority to search the user space for user DNs.
 - The `cn=` is the *common name* of the security principal.
- The `ldap.searchAuthPW` property specifies password Splice Machine should use to perform the DN search.
- The `ldap.searchBase` property specifies the root DN of the point in your hierarchy from which to begin a guest or anonymous search for the user's DN.

- The `ldap.searchFilter` property specifies the search filter to use to determine what constitutes a user while searching for a user DN

Connecting with JDBC and LDAP

You can then use our JDBC driver to connect to your database with LDAP authentication, using a connection string similar to this:

```
jdbc:splice://localhost:1527/splicedb;user=yourName;password=yourPswd
```

Configuring SSL/TLS Authentication

This is an On-Premise-Only topic! [Learn about our products](#)

This topic describes how to configure SSL/TLS on your cluster to support secure JDBC connections to your Splice Machine database.

By default, JDBC connections to your database are not encrypted. You can configure SSL/TLS authentication for two different encryption modes:

- *Basic SSL/TLS Encryption* encrypts the data sent back and forth between a client and the server.
- *SSL/TLS Encryption with Peer Authentication* encrypts the data sent between client and server, and adds a layer of authentication known as peer authentication, which uses trusted certificates to authenticate the sender and/or receiver.

The term *peer* is used in this context to refer to the other side of a server-client communication: the client is the server's peer, and the server is the client's peer. You can set up peer authentication on a server, a client, or both.

The remainder of this topic shows you how to configure this authentication, in these sections:

- » [Generating Certificates](#) walks you through generating the certificates.
- » [Importing Certificates](#) describes how to import certificates to clients and servers.
- » [Updating Configuration Options](#) shows you how to update your server's configuration options and then restart your server.
- » [Restarting your Server](#) describes how to restart the server so that your updated security options take effect.
- » [Connecting Securely From a Client](#) walks you through connecting securely from various clients.

NOTE: We use **highlighted text** in this section to display sample names that you should replace with your own names.

Generating Certificates

This section shows you how to generate the required, trusted certificates on your server and client(s).



To configure your database for SSL/TLS authentication, you'll need to use the *keytool* application that is included with the JDK; this tool is documented here: <http://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html>.

Generate a Server Certificate

Use the *keytool* to generate a key-pair in the keystore. Here's an example of interacting with the *keytool*:

```
% keytool -genkey -alias MyServerName -keystore ~/vault/ServerKeyStore
Enter keystore password: myPassword
Re-enter new password: myPassword
What is your first and last name?
[Unknown]: John Doe
What is the name of your organizational unit?
[Unknown]: TechPubs
What is the name of your organization?
[Unknown]: MyCompany
What is the name of your City or Locality?
[Unknown]: San Francisco
What is the name of your State or Province?
[Unknown]: CA
What is the two-letter country code for this unit?
[Unknown]: US
IS CN=John Doe, OU=TechPubs, O=MyCompany, L=San Francisco, ST=CA, C=US correct?
[no]: yes

Enter key password for <MyServerName>
(RETURN if same as keystore password): myPassword
```

Now issue this *keytool* command to generate the certificate from the key you just created:

```
% keytool -export -alias MyServerName \
> -keystore ~/value/ServerKeyStore -rfc -file ServerCertificate \
> -storepass myPassword
Certificate stored in file <ServerCertificate>
% ls -ltr
total 8
-rw-rw-r-- 1 myName myGroup 1295 Aug 3 23:15 ServerKeyStore
-rw-rw-r-- 1 myName myGroup 1181 Aug 3 23:23 ServerCertificate
%
```

Generate Client Certificates

Now use the *keytool* to generate a client key-pair in the keystore; for example:

```
% keytool -genkey -alias MyClientName -keystore ~/vault/ClientKeyStore
```

Respond to the questions in the same way as you did when generating the server key-pair.

And then generate the client certificate:

```
% keytool -export -alias MyClientName \
> -keystore ~/value/ClientKeyStore -rfc -file ClientCertificate \
> -storepass myPassword
Certificate stored in file <ClientCertificate>
% ls -ltr
total 8
-rw-rw-r-- 1 myName myGroup 1295 Aug 3 23:15 ClientKeyStore
-rw-rw-r-- 1 myName myGroup 1181 Aug 3 23:23 ClientCertificate
%
```

Importing Certificates

After you've generated your certificates, it's time to:

- » [Import the server certificate to the client keystore.](#)
- » [Import the client certificate to the server keystore.](#)
- » [Secure and deploy the keystore and trust store.](#)

Import Server Certificate to Client Keystore

You need to copy (scp) the server certificate to the client, and then use a *keytool* command like this to import the certificate:

```
% keytool -import -alias favoriteServerCertificate \
-file ServerCertificate -keystore ~/vault/ClientTrustStore \
-storepass secretClientTrustStorePassword
Owner: CN=John Doe, OU=TechPubs, O=MyCompany, L=San Francisco, ST=CA, C=US
Issuer: CN=John Doe, OU=TechPubs, O=MyCompany, L=San Francisco, ST=CA, C=US
Serial number: 24a2c7f7
Valid from: Thu Aug 03 23:15:52 UTC 2017 until: Wed Nov 01 23:15:52 UTC 2017
Certificate fingerprints:
      MD5: 5B:86:E9:C9:DF:E1:34:41:C8:B0:55:DE:C6:34:8A:21
      SHA1: 17:C8:43:FE:9C:FF:0C:4A:B2:51:36:0C:79:16:EB:73:24:C3:5A:83
      SHA256: 8D:55:1A:10:37:39:21:14:8E:21:3A:10:78:A1:C7:25:5F:9C:A7:8D:4E:3F:87:4
0:A0:ED:70:BE:EC:0F:7A:D9
      Signature algorithm name: SHA1withDSA
      Version: 3
```

Extensions:

```
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 8B 71 1E 04 E7 E4 84 E6      35 B3 6B EB B5 92 1A 35  .q.....5.k....5
0010: 5E FD B1 40                                         ^...@]
]
```

Import Client Certificates to Server Keystore

Next, copy (`scp`) the client certificate to the region server and use `keytool` to import the certificate:

```
% keytool -import -alias Client_1_Certificate \
-file ClientCertificate -keystore ~/vault/ServerTrustStore \
-storepass secretServerTrustStorePassword
Owner: CN=John Doe, OU=TechPubs, O=MyCompany, L=San Francisco, ST=CA, C=US
Issuer: CN=John Doe, OU=TechPubs, O=MyCompany, L=San Francisco, ST=CA, C=US
Serial number: 7507d351
Valid from: Thu Aug 03 23:25:19 UTC 2017 until: Wed Nov 01 23:25:19 UTC 2017
Certificate fingerprints:
      MD5: 19:46:4C:D5:6C:A5:40:AC:6C:F6:2C:DC:7B:86:C5:45
      SHA1: BB:D7:9C:5E:5A:EB:E3:D1:F0:22:49:47:D4:C5:31:24:1E:06:0F:AE
      SHA256: 88:8E:6E:97:ED:78:B1:AE:5E:65:09:30:C2:E8:AF:B3:DD:40:5A:7B:19:97:ED:0
4:E0:A3:82:66:E9:A4:3E:2A
      Signature algorithm name: SHA1withDSA
      Version: 3

Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 59 37 E4 92 34 0A A2 45    93 E6 45 3A AF 57 77 E8  Y7..4..E..E:.Ww.
0010: E6 B9 24 08                           ...$.
]
]

Trust this certificate? [no]: yes
Certificate was added to keystore
%
```

Secure and Deploy the Keystore and Trust Store

Once you've imported your certificates, you should copy the `vault` directory to a location that is directly accessible from HBase. We recommend copying it to a directory such as `/etc/vault` or `/hbase/vault`.



You **MUST** deploy the `vault` directory to the same location on each region server, so that all region servers can access it during server startup.

Updating Configuration Options

Now that you've got your certificates all set up, you need to modify a few configuration options and restart the server to take your new security options live. On a CDH cluster, you need to update the region server Java options, which you'll find in the Admin console:

CDH->HBase->Configuration->Java Configuration Options for HBase Region Server

- » If you're using basic SSL/TLS (without peer authentication), add this property:

```
-Dderby.drda.sslMode=basic
```

- » If you're using full SSL/TLS (with peer authentication), add these properties:

```
-Dderby.drda.sslMode=peerAuthentication
-Djavax.net.ssl.keyStore=/tmp/vault/ServerKeyStore
-Djavax.net.ssl.keyStorePassword=myPassword
-Djavax.net.ssl.trustStore=/tmp/vault/ServerTrustStore
-Djavax.net.ssl.trustStorePassword=secretServerTrustStorePassword
```

Rebooting Your Cluster

Once you've updated your configuration, restart HBase to make the changes effective in your cluster. After HBase restart, you can verify that the server started in secure mode by examining the logs: If you look at /var/log/hbase/splice.log, you should see a message similar to this:

```
Mon AUG 28 04:52:03 UTC 2017 : Splice Machine Network Server - 10.9.2.2 - (1) started and ready to accept SSL connections on port 1527
```

Connecting Securely From a Client

You can now connect securely to your database. This section provides several examples:

- » [Running the splice> Command Line Securely](#)
- » [Running a JDBC Client App Securely](#)
- » [Adding New Client Nodes](#)
- » [Connecting Securely From a Third Party Client](#)

Running the splice> Command Line Securely

To run the splice> command line securely, you need to export several env variables before starting sqlshell. You can then issue a connect command that specifies the type of security, as shown below.

First export the environmental variables that specify your key stores:

```
export CLIENT_SSL_KEYSTORE=/home/splice/vault/ClientKeyStore
export CLIENT_SSL_KEYSTOREPASSWD=myPassword
export CLIENT_SSL_TRUSTSTORE=/home/splice/vault/ClientTrustStore
export CLIENT_SSL_TRUSTSTOREPASSWD=secretClientTrustStorePassword
```

Then, start the splice> command line:

```
% ./sqlshell.sh
```

The `sqlshell` command issues a default (no security) connection command. To connect securely, add an `ssl=` option to the `connect` command. You use different `connect` commands for each of the three security modes:

Security Mode	Connect Command
None	<code>connect 'jdbc:splice://x.x.x.xxx:1527/splicedb;user=splice;password=admin';</code>
Basic SSL	<code>connect 'jdbc:splice://x.x.x.xxx:1527/spicedb;user=splice;password=admin;ssl=basic';</code>
SSL w/Peer Authentication	<code>connect 'jdbc:splice://x.x.x.xxx:1527/spicedb;user=splice;password=admin;ssl=peerAuthentication';</code>

Running a JDBC Client App Securely

To use a secured connection with a JDBC client app, you need to specify a connection string that includes the `ssl` option. If you don't specify this option, the default JDBC connection is unsecured, as shown in the *Connect Command* table in the previous section.

Here's a sample declaration for a peer authenticated connection to a Splice Machine database:

```
String dbUrl = "jdbc:splice://1.2.3.456:1527/spicedb;user=splice;password=admin;ssl=peerAuthentication";
```

We can create a Java program that includes that declaration and then compile it into `SampleJDBC.java` with a command like this:

```
+ javac -classpath "../db-client-2.6.0.1729-SNAPSHOT.jar" ./SampleJDBC.java
```

We can then use a command like this to execute and JDBC app with the correct SSL keystore and truststore properties:

```
% java -classpath ../db-client-2.6.0.1729-SNAPSHOT.jar
-Djavax.net.ssl.keyStore=/home/splice/vault/ClientKeyStore
-Djavax.net.ssl.keyStorePassword=myPassword
-Djavax.net.ssl.trustStore=/home/splice/vault/ClientTrustStore
-Djavax.net.ssl.trustStore.ssl.trustStorePassword=secretClientTrustStorePassword
SampleJDBC
```

Adding New Client Nodes

Whenever you connect a new client node to a server, you need to perform a few steps to enable SSL/TLS on the new node:

- » [Generate a new client certificate.](#)
- » [Import the new client certificate into the server's keystore.](#)
- » [Import the server certificate into the new client's keystore.](#)
- » Restart the server.

Finally, you need to set these env variables:

```
export CLIENT_SSL_KEYSTORE=/home/splice/vault/ClientKeyStore
export CLIENT_SSL_KEYSTOREPASSWD=myPassword
export CLIENT_SSL_TRUSTSTORE=/home/splice/vault/ClientTrustStore
export CLIENT_SSL_TRUSTSTOREPASSWD=secretClientTrustStorePassword
```

Connecting Securely From a Third Party Client

This section describes what you need to do to connect securely to your Splice Machine from a third party client. We use Zeppelin as an example; other clients will have similar requirements. For Zeppelin, follow these steps

1. Navigate to and edit the `bin/interpreter.sh` file in the `Zeppelin` installation directory.
2. Find the `JAVA_INTP_OPTS` property definition.
3. Append the following SSL properties onto that definition:

```
JAVA_INTP_OPTS+="
-Dzeppelin.log.file=${ZEPPELIN_LOGFILE} \
-Djavax.net.ssl.keyStore=${CLIENT_SSL_KEYSTORE} \
-Djavax.net.ssl.keyStorePassword=${CLIENT_SSL_KEYSTOREPASSWD} \
-Djavax.net.ssl.trustStore=${CLIENT_SSL_TRUSTSTORE} \
-Djavax.netDjavax.net.ssl.trustStore.ssl.trustStorePassword=${CLIENT_SSL_TRUSTSTORE_PASSWD}"
```

4. Make sure that you have exported the SSL keystore and truststore env variables:

```
export CLIENT_SSL_KEYSTORE=/home/splice/vault/ClientKeyStore
export CLIENT_SSL_KEYSTOREPASSWD=myPassword
export CLIENT_SSL_TRUSTSTORE=/home/splice/vault/ClientTrustStore
export CLIENT_SSL_TRUSTSTOREPASSWD=secretClientTrustStorePassword
```

5. Restart Zeppelin:

```
% zeppelin-daemon.sh start
```

6. Create a new JDBC Interpreter

Navigate to the [Zeppelin interface URL](#), then click **Interpreter->+Create** to create a new interpreter. The image below shows sample settings for the new interpreter:

test_peer %test_peer •**Option**

The interpreter will be instantiated in process.

- Connect to existing process
- Set permission

Properties

name	value
common.max_count	1000
default.driver	com.splicemachine.db.jdbc.ClientDriver
default.password	admin
default.url	jdbc:splice://localhost:1527/splicedb;ssl=peerAuthentication
default.user	splice
zeppelin.interpreter.localRepo	/Users/ xxxxxx /zeppelin-0.7.2-bin-all/local-repo/2CS4VNH1P
zeppelin.interpreter.output.limit	102400
zeppelin.jdbc.auth.type	
zeppelin.jdbc.concurrent.max_connection	10
zeppelin.jdbc.concurrent.use	true
zeppelin.jdbc.keytab.location	
zeppelin.jdbc.principal	

Dependencies

artifact	exclude
/Users/ xxxxxx /dbc-test/db-client-2.6.1.1732-SNAPSHOT.jar	



Be sure to provide the correct JDBC driver location in the artifact dependencies section.

7. Save the new interpreter.
8. Create a new Note with the new interpreter.

Using HAProxy with Splice Machine on a Kerberos-Enabled Cluster

This topic shows you how JDBC and ODBC applications can authenticate to the backend region server through HAProxy on a Splice Machine cluster that has Kerberos enabled.



Our Developer's Guide includes the Connecting to Splice Machine Through HAProxy topic, which shows you how to configure HAProxy for use with Splice Machine.

Configuring Kerberos on CDH

You can enable Kerberos mode on a CDH5.8.x or later cluster using the configuration wizard described here:

https://www.cloudera.com/documentation/enterprise/5-8-x/topics/cm_sg_intro_kerb.html

Kerberos and Application Access

As a Kerberos pre-requisite for Splice Machine JDBC and ODBC access:

- » Database users must be added in the Kerberos realm as principals
- » Keytab entries must be generated and deployed to the remote clients on which the applications are going to connect.

HAProxy will then transparently forward the connections to the back-end cluster in Kerberos setup.

Kerberos and ODBC Access

To connect to a Kerberos-enabled cluster with ODBC, follow these steps:

1. Verify that the odbc.ini configuration file for the DSN you're connecting to includes this setting:

```
USE_KERBEROS=1
```

See our Using the Splice Machine ODBC Driver for more information.

2. A default security principal user must be established with a TGT in the ticket cache prior to invoking the driver. You can use the following command to establish the principal user:

```
kinit principal
```

Where *principal* is the name of the user who will be accessing Splice Machine. Enter the password for this user when prompted.

3. Launch the application that will connect using ODBC. The ODBC driver will use that default Kerberos *principal* when authenticating with Splice Machine.

Example

This example assumes that you are using the default user name `splice`. Follow these steps to connect with through HAProxy:

- 1. Create the principal in Kerberos Key Distribution Center**

Create the principal `splice@kerberos_realm_name` in Kerberos Key Distribution Center (KDC). This generates a keytab file named `splice.keytab`.

- 2. Copy the generated keytab file**

Copy the `splice.keytab` file that to all client systems.

- 3. Connect:**

You can now connect to the Kerberos-enabled Splice Machine cluster with JDBC through HAProxy, using the following URL:

```
jdbc:splice://<haproxy_host>:1527/splicedb;principal=splice@<realm_name>;keytab=/<path>/splice.keytab
```

Use the same steps to allow other Splice Machine users to connect by adding them to the Kerberos realm and copying the keytab files to their client systems. This example sets up access for a new user name `jdoe`.

- 1. Create the user in your Splice Machine database:**

```
call syscs_util.syscs_create_user( 'jdoe', 'jdoe' );
```

- 2. Grant privileges to the user**

For this example, we are granting all privileges on a table named `myTable` to the new user:

```
grant all privileges on splice.myTable to jdoe;
```

- 3. Use KDC to create a new principal and generate a keytab file. For example:**

```
# kadmin.local addprinc -randkey jdoe@SPLICEMACHINE.COLO
```

- 4. Set the password for the new principal:**

```
# kadmin.local cpw jdoeEnter password for principal "jdoe@SPLICEMACHINE.COLO":
```

- 5. Create keytab file `jdoe.keytab`**

```
# kadmin: xst -k jdoe.keytab jdoe@SPLICEMACHINE.COLO
```

6. Copy the generated keytab file to the client system**7. Connect through HAProxy with the following URL:**

```
jdbc:splice://ha-proxy-host:1527/splicedb;principal=jdoe@SPLICEMACHINE.CO  
LO;keytab=/home/splice/user1.keytab
```

Importing Data into Splice Machine

This section walks you through importing data into Splice Machine, in these topics:

- » Overview of Importing Data
- » Import Parameters
- » Handling Import Input Data
- » Import Error Handling
- » Examples of Importing Data
- » Examples of Using Bulk HFile Import
- » Example: Importing TPCH Benchmark Data

Importing Data Into Your Splice Machine Database

This tutorial guides you through importing (loading) data into your Splice Machine database. It contains these topics:

Tutorial Topic	Description
1: Tutorial Overview	<i>This topic.</i> Introduces the import options that are available to you and helps you determine which option best meets your needs.
2: Parameter Usage	Provides detailed specifications of the parameter values you must supply to the import procedures.
3: Input Data Handling	Provides detailed information and tips about input data handling during ingestion.
4: Error Handling	Helps you to understand and use logging to discover and repair any input data problems that occur during an ingestion process.
5: Usage Examples	Walks you through examples of importing data with the <code>SYSCS_UTIL.IMPORT_DATA</code> , <code>SYSCS_UTIL.UPSERT_DATA_FROM_FILE</code> , and <code>SYSCS_UTIL.MERGE_DATA_FROM_FILE</code> system procedures.
6: Bulk HFile Examples	Walks you through examples of using the <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> system procedure.
7: Importing TPCH Data	Walks you through importing TPCH sample data into your database.

Overview of Importing Data Into Your Database

The remainder of this topic introduces you to importing (loading) data into your Splice Machine database. It summarizes the different import procedures we provide, presents a quick look at the procedure declarations, and helps you to decide which one matches your conditions. It contains the following sections:

- » [Data Import Procedures](#) summarizes the built-in system procedures that you can use to import your data.
- » [Which Procedure Should I Use to Import My Data?](#) presents a decision tree that makes it easy to decide which procedure to use for your circumstances.
- » [Import Procedures Syntax](#) shows the syntax for our import procedures.

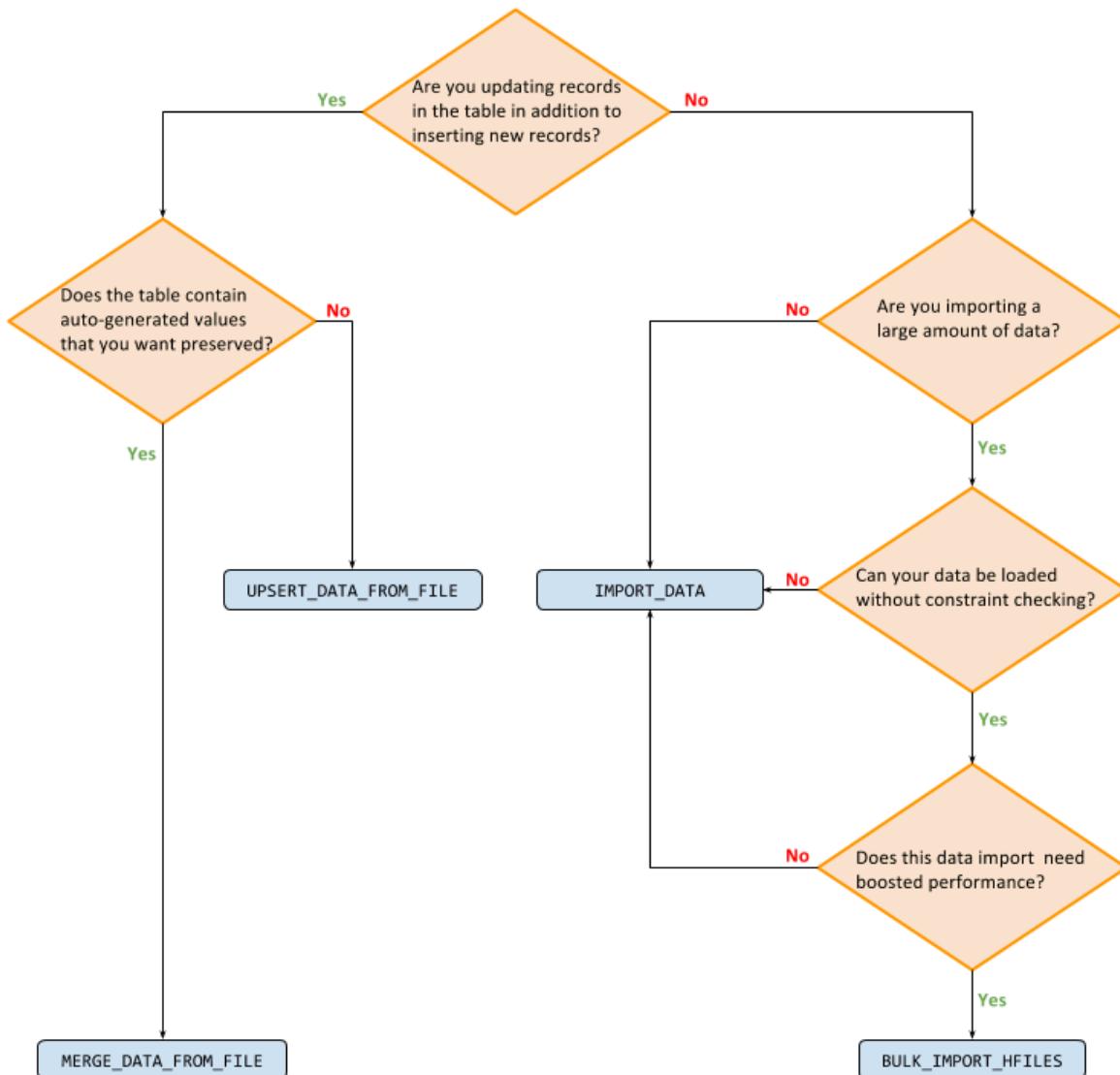
Data Import Procedures

Splice Machine includes four different procedures for importing data into your database, three of which use identical syntax; the fourth provides a more behind-the-scenes method that is quicker when loading large data sets, but requires more work and care on your part. The table below summarizes these import procedures:

System Procedure	Description
SYSCS_UTIL.IMPORT_DATA	Imports data into your database, creating a new record in your table for each record in the imported data. <code>SYSCS_UTIL.IMPORT_DATA</code> inserts the default value of each column that is not specified in the input.
SYSCS_UTIL.UPSERT_DATA_FROM_FILE	Imports data into your database, creating new records and *updating existing records* in the table. Identical to <code>SYSCS_UTIL.IMPORT_DATA</code> except that will update matching records. <code>SYSCS_UTIL.UPSERT_DATA_FROM_FILE</code> also inserts or updates the value in the table of each column that is not specified in the input; inserting the default value (or NULL if there is no default) for that column.
SYSCS_UTIL.MERGE_DATA_FROM_FILE	Imports data into your database, creating new records and *updating existing records* in the table. Identical to <code>SYSCS_UTIL.UPSERT_DATA_FROM_FILE</code> except that it does not replace values in the table for unspecified columns when updating an existing record in the table.
SYSCS_UTIL.BULK_IMPORT_HFILE	Takes advantage of HBase bulk loading to import table data into your database by temporarily converting the table file that you're importing into HFiles, importing those directly into your database, and then removing the temporary HFiles. This procedure uses syntax very similar to the other import procedures and has improved performance for large tables; however, the bulk HFile import requires extra work on your part and lacks constraint checking.

Which Procedure Should I Use to Import My Data?

The following diagram helps you decide which of our data importation procedures best fits your needs:



Notes

- » The `IMPORT_DATA` procedure imports new records into a database. The `UPSERT_DATA_FROM_FILE` and `MERGE_DATA_FROM_FILE` procedures import new records and update existing records. Importing all new records is faster because the database doesn't need to check if the record already exists in the database.
- » If your table contains auto-generated column values and you don't want those values overwritten when a record gets updated, use the `MERGE_DATA_FROM_FILE` procedure (`UPSERT_DATA_FROM_FILE` will overwrite).
- » The `BULK_IMPORT_HFILE` procedure is great when you're importing a very large dataset and need extra performance. However, it does not perform constraint checking.

Import Procedures Syntax

Here are the declarations of our four data import procedures; as you can see, three of our four import procedures use identical parameters, and the fourth (SYSCS_UTIL.BULK_IMPORT_HFILE) adds a couple extra parameters at the end:

```
call SYSCS_UTIL.IMPORT_DATA (
    schemaName,
    tableName,
    insertColumnList | null,
    fileOrDirectoryName,
    columnDelimiter | null,
    characterDelimiter | null,
    timestampFormat | null,
    dateFormat | null,
    timeFormat | null,
    badRecordsAllowed,
    badRecordDirectory | null,
    oneLineRecords | null,
    charset | null
);
```

```
call SYSCS_UTIL.UPSERT_DATA_FROM_FILE (
    schemaName,
    tableName,
    insertColumnList | null,
    fileOrDirectoryName,
    columnDelimiter | null,
    characterDelimiter | null,
    timestampFormat | null,
    dateFormat | null,
    timeFormat | null,
    badRecordsAllowed,
    badRecordDirectory | null,
    oneLineRecords | null,
    charset | null
);
```

```
call SYSCS_UTIL.MERGE_DATA_FROM_FILE (
    schemaName,
    tableName,
    insertColumnList | null,
    fileOrDirectoryName,
    columnDelimiter | null,
    characterDelimiter | null,
    timestampFormat | null,
    dateFormat | null,
    timeFormat | null,
    badRecordsAllowed,
    badRecordDirectory | null,
    oneLineRecords | null,
    charset | null
);
```

```
call SYSCS_UTIL.BULK_IMPORT_HFILE (
    schemaName,
    tableName,
    insertColumnList | null,
    fileName,
    columnDelimiter | null,
    characterDelimiter | null,
    timestampFormat | null,
    dateFormat | null,
    timeFormat | null,
    maxBadRecords,
    badRecordDirectory | null,
    oneLineRecords | null,
    charset | null,
    bulkImportDirectory,
    skipSampling
);
```

You'll find descriptions and detailed reference information for all of these parameters in the Import Parameters topic of this tutorial.

And you'll find detailed reference descriptions of all four procedures in our SQL Reference Manual.

See Also

- » [Importing Data: Input Parameters](#)
- » [Importing Data: Input Data Handling](#)
- » [Importing Data: Error Handling](#)
- » [Importing Data: Usage Examples](#)
- » [Importing Data: Bulk HFile Examples](#)

- » Importing Data: Importing TPCH Data
- » [SYSCS_UTIL.IMPORT_DATA](#)
- » [SYSCS_UTIL.UPSERT_DATA_FROM_FILE](#)
- » [SYSCS_UTIL.MERGE_DATA_FROM_FILE](#)
- » [SYSCS_UTIL.BULK_IMPORT_HFILE](#)

Importing Data: Specifying the Import Parameter Values

This topic first shows you the syntax of each of the four import procedures, and then provides detailed information about the input parameters you need to specify when calling one of the Splice Machine data ingestion procedures.

Import Procedures Syntax

Three of our four data import procedures use identical parameters:

```
SYSCS_UTIL.IMPORT_DATA (
SYSCS_UTIL.UPSERT_DATA_FROM_FILE (
SYSCS_UTIL.MERGE_DATA_FROM_FILE (
    schemaName,
    tableName,
    insertColumnList | null,
    fileOrDirectoryName,
    columnDelimiter | null,
    characterDelimiter | null,
    timestampFormat | null,
    dateFormat | null,
    timeFormat | null,
    badRecordsAllowed,
    badRecordDirectory | null,
    oneLineRecords | null,
    charset | null
);
```

The fourth procedure, `SYSCS_UTIL.BULK_IMPORT_HFILE`, adds a couple extra parameters at the end:

```
SYSCS_UTIL.BULK_IMPORT_HFILE
( schemaName,
  tableName,
  insertColumnList | null,
  fileName,
  columnDelimiter | null,
  characterDelimiter | null,
  timestampFormat | null,
  dateFormat | null,
  timeFormat | null,
  maxBadRecords,
  badRecordDirectory | null,
  oneLineRecords | null,
  charset | null,
  bulkImportDirectory,
  skipSampling
);
```

Overview of Parameters Used in Import Procedures

All of the Splice Machine data import procedures share a number of parameters that describe the table into which you're importing data, a number of input data format details, and how to handle problematic records.

The following table summarizes these parameters. Each parameter name links to its reference description, found below the table:

Category	Parameter	Description	Example Value
Table Info	schemaName	The name of the schema of the table in which to import.	SPLICE
	tableName	The name of the table in which to import	playerTeams
Data Location	insertColumnList	The names, in single quotes, of the columns to import. If this is null, all columns are imported.	'ID, TEAM'
	fileOrDirectoryName	<p>Either a single file or a directory. If this is a single file, that file is imported; if this is a directory, all of the files in that directory are imported. You can import compressed or uncompressed files.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>The SYSCS_UTIL.MERGE_DATA_FROM_FILE procedure only works with single files; you cannot specify a directory name when calling SYSCS_UTIL.MERGE_DATA_FROM_FILE.</p> </div> <p>On a cluster, the files to be imported MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p>	/data/mydata/mytable.csv 's3a://splice-benchmark-data/flat/TPCH/100/region'
Data Formats	oneLineRecords	A Boolean value that specifies whether (true) each record in the import file is contained in one input line, or (false) if a record can span multiple lines.	true
	charset	The character encoding of the import file. The default value is UTF-8.	null
	columnDelimiter	The character used to separate columns, Specify null if using the comma (,) character as your delimiter.	' '
	characterDelimiter	The character is used to delimit strings in the imported data.	'''

Category	Parameter	Description	Example Value
	timestampFormat	<p>The format of timestamps stored in the file. You can set this to <code>null</code> if there are no time columns in the file, or if the format of any timestamps in the file match the Java.sql.Timestamp default format, which is: "yyyy-MM-dd HH:mm:ss".</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <p>All of the timestamps in the file you are importing must use the same format.</p> </div>	'yyyy-MM-dd HH:mm:ss.SSS'
	dateFormat	<p>The format of datestamps stored in the file. You can set this to <code>null</code> if there are no date columns in the file, or if the format of any dates in the file match pattern: "yyyy-MM-dd".</p>	yyyy-MM-dd
	timeFormat	<p>The format of time values stored in the file. You can set this to <code>null</code> if there are no time columns in the file, or if the format of any times in the file match pattern: "HH:mm:ss".</p>	HH:mm:ss
Problem Logging	badRecordsAllowed	<p>The number of rejected (bad) records that are tolerated before the import fails. If this count of rejected records is reached, the import fails, and any successful record imports are rolled back. Specify 0 to indicate that no bad records are tolerated, and specify -1 to indicate that all bad records should be logged and allowed.</p>	25
	badRecordDirectory	<p>The directory in which bad record information is logged. Splice Machine logs information to the <code><import_file_name>.bad</code> file in this directory; for example, bad records in an input file named <code>foo.csv</code> would be logged to a file named <code>badRecordDirectory/foo.csv.bad</code>.</p> <p>On a cluster, this directory MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p>	'importErrsDir'

Category	Parameter	Description	Example Value
Bulk HFile Import	bulkImportDirectory (outputDirectory)	<p>For <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code>, this is the name of the directory into which the generated HFiles are written prior to being imported into your database.</p> <p>For the <code>SYSCS_UTIL.COMPUTE_SPLIT_KEY</code> procedure, where it is named <code>outputDirectory</code>, this parameter specifies the directory into which the split keys are written.</p>	hdfs:///tmp/test_hfile_import/
	skipSampling	<p>The <code>skipSampling</code> parameter is a Boolean value that specifies how you want the split keys used for the bulk HFile import to be computed. Set to <code>false</code> to have <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> automatically determine splits for you.</p> <p>This parameter is only used with the <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> system procedure.</p>	<code>false</code>

Import Parameters Reference

This section provides reference documentation for all of the data importation parameters.

schemaName

The `schemaName` is a string that specifies the name of the schema of the table into which you are importing data.

Example: `SPLICEDB`

tableName

The `tableName` is a string that specifies the name of the table into which you are importing data.

Example: `playerTeams`

insertColumnList

The `insertColumnList` parameter is a string that specifies the names, in single quotes, of the columns you wish to import. If this is `null`, all columns are imported.

- » If you don't specify an `insertColumnList` and your input file contains more columns than are in the table, then the extra columns at the end of each line in the input file **are ignored**. For example, if your table contains columns (`a`, `b`, `c`) and your file contains columns (`a`, `b`, `c`, `d`, `e`), then the data in your file's `d` and `e` columns will be ignored.

- » If you do specify an `insertColumnList`, and the number of columns doesn't match your table, then any other columns in your table will be replaced by the default value for the table column (or `NULL` if there is no default for the column). For example, if your table contains columns (`a`, `b`, `c`) and you only want to import columns (`a`, `c`), then the data in table's `b` column will be replaced with the default value for that column.

Example: `ID, TEAM`

See *Importing and Updating Records* for additional information about handling of missing, generated, and default values during data importation.

fileOrDirectoryName

The `fileOrDirectoryName` (or `fileName`) parameter is a string that specifies the location of the data that you're importing. This parameter is slightly different for different procedures:

- » For the `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` or `SYSCS_UTIL.UPSERT_DATA_FROM_DIRECTORY` procedures, this is either a single file or a directory. If this is a single file, that file is imported; if this is a directory, all of the files in that directory are imported.
- » For the `SYSCS_UTIL.MERGE_DATA_FROM_FILE` and `SYSCS_UTIL.BULK_IMPORT_HFILE` procedure, this can only be a single file (directories are not allowed).

NOTE:

On a cluster, the files to be imported **MUST be on S3, HDFS (or MapR-FS)**, as must the `badRecordDirectory` directory. If you're using our Database Service product, files can only be imported from S3. The files must also be readable by the `hbase` user.

Example: `data/mydata/mytable.csv`

Importing from S3

If you are importing data that is stored in an S3 bucket on AWS, you need to specify the data location in an `s3a` URL that includes access key information.

Example: `s3a://splice-benchmark-data/flat/TPCH/100/region`

See *Specifying Your Input Data Location* for additional information about specifying your input data location.

Importing Compressed Files

Note that files can be compressed or uncompressed, including BZIP2 compressed files.

Importing multiple files at once improves parallelism, and thus speeds up the import process. Uncompressed files can be imported faster than compressed files. When using compressed files, the compression algorithm makes a difference; for example,

- » gzip-compressed files cannot be split during importation, which means that import work on such files cannot be performed in parallel.
- » In contrast, bzip2-compressed files can be split and thus can be imported using parallel tasks. Note that bzip2 is CPU intensive compared to LZ4 or LZ0, but is faster than gzip because files can be split.

oneLineRecords

The oneLineRecords parameter is a Boolean value that specifies whether each line in the import file contains one complete record:

- » If you specify `true` or `null`, then each record is expected to be found on a single line in the file.
- » If you specify `false`, records can span multiple lines in the file.

Multi-line record files are slower to load, because the file cannot be split and processed in parallel; if you import a directory of multiple line files, each file as a whole is processed in parallel, but no splitting takes place.

Example: `true`

charset

The charset parameter is a string that specifies the character encoding of the import file. The default value is `UTF-8`.

NOTE: Currently, any value other than `UTF-8` is ignored, and `UTF-8` is used.

Example: `null`

columnDelimiter

The columnDelimiter parameter is a string that specifies the character used to separate columns. You can specify `null` if using the comma (,) character as your delimiter.

In addition to using plain text characters, you can specify the following special characters as delimiters:

Special character	Display
\t	Tab

Special character	Display
\f	Formfeed
\b	Backspace
\\"	Backslash
^a (or ^A)	Control-a

NOTE: If you are using a script file from the `splice>` command line, your script can contain the actual Control-a character as the value of this parameter.

Example: ' | '

See *Column Delimiters* for additional information about column delimiters.

characterDelimiter

The `characterDelimiter` parameter is a string that specifies which character is used to delimit strings in the imported data. You can specify null or the empty string to use the default string delimiter, which is the double-quote ("").

In addition to using plain text characters, you can specify the following special characters as delimiters:

Special character	Display
\t	Tab
\f	Formfeed
\b	Backspace
\\"	Backslash

Special character	Display
^a (or ^A)	Control-a <p>NOTE: If you are using a script file from the <code>splice></code> command line, your script can contain the actual Control-a character as the value of this parameter.</p>

Notes:

- » If your input contains control characters such as newline characters, make sure that those characters are embedded within delimited strings.
- » To use the single quote (') character as your string delimiter, you need to escape that character. This means that you specify four quotes (' ' ' ') as the value of this parameter. This is standard SQL syntax.

Example: ''

See *Character Delimiters* for additional information about character delimiters.

timestampFormat

The `timestampFormat` parameter specifies the format of timestamps in your input data. You can set this to `null` if either:

- » there are no time columns in the file
- » all time stamps in the input match the `Java.sql.Timestamp` default format, which is: "yyyy-MM-dd HH:mm:ss".



All of the timestamps in the file you are importing must use the same format.

Splice Machine uses the following Java date and time pattern letters to construct timestamps:

Pattern Letter	Description	Format(s)
y	year	yy or yyyy
M	month	MM
d	day in month	dd

Pattern Letter	Description	Format(s)
h	hour (0-12)	hh
H	hour (0-23)	HH
m	minute in hour	mm
s	seconds	ss
S	tenths of seconds	S, SS, SSS, SSSS, SSSSS or SSSSSS*
		*Specify SSSSSS to allow a variable number (any number) of digits after the decimal point.
z	time zone text	e.g. Pacific Standard time
Z	time zone, time offset	e.g. -0800

The default timestamp format for Splice Machine imports is: yyyy-MM-dd HH:mm:ss, which uses a 24-hour clock, does not allow for decimal digits of seconds, and does not allow for time zone specification.

NOTE: The standard Java library does not support microsecond precision, so you **cannot** specify millisecond (S) values in a custom timestamp format and import such values with the desired precision.

Timestamps and Importing Data at Different Locations

Note that timestamp values are relative to the geographic location at which they are imported, or more specifically, relative to the timezone setting and daylight saving time status where the data is imported.

This means that timestamp values from the same data file may appear differently after being imported in different timezones.

Examples

The following tables shows valid examples of timestamps and their corresponding format (parsing) patterns:

Timestamp value	Format Pattern	Notes
2013-03-23 09:45:00	yyyy-MM-dd HH:mm:ss	This is the default pattern.
2013-03-23 19:45:00.98-05	yyyy-MM-dd HH:mm:ss.SSZ	This pattern allows up to 2 decimal digits of seconds, and requires a time zone specification.

Timestamp value	Format Pattern	Notes
2013-03-23 09:45:00-07	yyyy-MM-dd HH:mm:ssZ	This pattern requires a time zone specification, but does not allow for decimal digits of seconds.
2013-03-23 19:45:00.98-0530	yyyy-MM-dd HH:mm:ss.SSZ	This pattern allows up to 2 decimal digits of seconds, and requires a time zone specification.
2013-03-23 19:45:00.123	yyyy-MM-dd HH:mm:ss.SSS	This pattern allows up to 3 decimal digits of seconds, but does not allow a time zone specification.
2013-03-23 19:45:00.12		Note that if your data specifies more than 3 decimal digits of seconds, an error occurs.
2013-03-23 19:45:00.1298	yyyy-MM-dd HH:mm:ss.SSSS	This pattern allows up to 4 decimal digits of seconds, but does not allow a time zone specification.

See *Time and Date Formats in Input Records* for additional information about date, time, and timestamp values.

dateFormat

The dateFormat parameter specifies the format of datestamps stored in the file. You can set this to null if either:

- » there are no date columns in the file
- » the format of any dates in the input match this pattern: "yyyy-MM-dd".

Example: yyyy-MM-dd

See *Time and Date Formats in Input Records* for additional information about date, time, and timestamp values.

timeFormat

The timeFormat parameter specifies the format of time values in your input data. You can set this to null if either:

- » there are no time columns in the file
- » the format of any times in the input match this pattern: "HH:mm:ss".

Example: HH:mm:ss

See *Time and Date Formats in Input Records* for additional information about date, time, and timestamp values.

badRecordsAllowed

The badRecordsAllowed parameter is integer value that specifies the number of rejected (bad) records that are tolerated before the import fails. If this count of rejected records is reached, the import fails, and any successful record imports are rolled back.

These values have special meaning:

- » If you specify -1 as the value of this parameter, all record import failures are tolerated and logged.
- » If you specify 0 as the value of this parameter, the import will fail if even one record is bad.

Example: 25

badRecordDirectory

The badRecordDirectory parameter is a string that specifies the directory in which bad record information is logged. The default value is the directory in which the import files are found.

Splice Machine logs information to the <import_file_name>.bad file in this directory; for example, bad records in an input file named foo.csv would be logged to a file named *badRecordDirectory*/foo.csv.bad.

The badRecordDirectory directory must be writable by the hbase user, either by setting the user explicitly, or by opening up the permissions; for example:

```
sudo -su hdfs hadoop fs -chmod 777 /badRecordDirectory
```

Example: 'importErrsDir'

bulkImportDirectory

NOTE: This parameter is only used with the `SYSCS_UTIL.BULK_IMPORT_HFILE` system procedure.

The bulkImportDirectory parameter is a string that specifies the name of the directory into which the generated HFiles are written prior to being imported into your database. The generated files are automatically removed after they've been imported.

Example: 'hdfs:///tmp/test_hfile_import/'

Please review the Bulk HFile Import Walkthrough topic to understand how importing bulk HFiles works.

skipSampling

NOTE: This parameter is only used with the `SYSCS_UTIL.BULK_IMPORT_HFILE` system procedure.

The `skipSampling` parameter is a Boolean value that specifies how you want the split keys used for the bulk HFile import to be computed:

- » If `skipSampling` is `true`, you need to use our [`SYSCS_UTIL.COMPUTE_SPLIT_KEY`](#) and [`SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS`](#) system procedures to manually split your table before calling `SYSCS_UTIL.BULK_IMPORT_HFILE`. This allows you more control over the splits, but adds a layer of complexity.
- » If `skipSampling` is `false`, then `SYSCS_UTIL.BULK_IMPORT_HFILE` samples your input data and computes the table splits for you, in the following steps. It:
 1. Scans (sample) the data
 2. Collects a rowkey histogram
 3. Uses that histogram to calculate the split key for the table
 4. Uses the calculated split key to split the table into HFiles

Example: `false`

Please review the Bulk HFile Import Walkthrough topic to understand how importing bulk HFiles works.

See Also

- » Importing Data: Tutorial Overview
- » Importing Data: Input Data Handling
- » Importing Data: Error Handling
- » Importing Data: Usage Examples
- » Importing Data: Bulk HFile Examples
- » Importing Data: Importing TPCH Data
- » [`SYSCS_UTIL.IMPORT_DATA`](#)
- » [`SYSCS_UTIL.UPSERT_DATA_FROM_FILE`](#)
- » [`SYSCS_UTIL.MERGE_DATA_FROM_FILE`](#)
- » [`SYSCS_UTIL.BULK_IMPORT_HFILE`](#)

Importing Data: Input Considerations

This topic provides detailed information about how the parameter values you specify when importing data are handled by Splice Machine's built-in import procedures.

For a summary of our import procedures and determining which to use, please see Importing Data: Overview.

For reference descriptions of the parameters used by those import procedures, please see Importing Data: Parameter Usage.

This topic includes the following sections:

Section	Description
Specifying Your Input Data Location	Describes how to specify the location of your input data when importing.
Input Data File Format	Information about input data files, including importing compressed files and multi-line records.
Delimiters in Your Input Data	Discusses the use of column and characters delimiters in your input data.
Time and Date Formats in Input Records	All about the date, time, and timestamp values in your input data.
Importing and Updating Records	Discusses importing new records and updating existing database records, handling missing values in the input data, and handling of generated and default values.
Importing CLOBs and BLOBs	Discussion of importing CLOBs and BLOBs into your Splice Machine database.
Scripting Your Imports	Shows you how to script your import processes.

Specifying Your Input Data Location

Some customers get confused by the the `fileOrDirectoryName` parameter that's used in our import procedures. How you use this depends on whether you are importing a single file or a directory of files, and whether you're importing data into a standalone version or cluster version of Splice Machine. This section contains these three subsections:

- » [Standalone Version Input File Path](#)
- » [HBase Input File Path](#)
- » [AWS Input File Path](#)

Standalone Version Input File Path

If you are running a stand alone environment, the name or path will be to a file or directory on the file system. For example:

```
/users/myname/mydata/mytable.csv/users/myname/mydatadir
```

HBase Input File Path

If you are running this on a cluster, the path is to a file on HDFS (or the MapR File system). For example:

```
/data/mydata/mytable.csv/data/myname/mydatadir
```

AWS S3 Input File Path

Finally, if you're importing data from an S3 bucket, you need to supply your AWS access and secret key codes, and you need to specify an s3a URL. This is also true for logging bad record information to an S3 bucket directory, as will be the case when using our Database-as-Service product.

For information about configuring Splice Machine access on AWS, please review our Configuring an S3 Bucket for Splice Machine Access topic, which walks you through using your AWS dashboard to generate and apply the necessary credentials.

Once you've established your access keys, you can include them inline; for example:

```
call SYSCS_UTIL.IMPORT_DATA ('TPCH', 'REGION', null, 's3a://(access key):(secret key)@splice-benchmark-data/flat/TPCH/100/region', '|', null, null, null, null, -1, 's3a://(access key):(secret key)@splice-benchmark-data/flat/TPCH/100/importLog', true, null);
```

Alternatively, you can specify the keys once in the `core-site.xml` file on your cluster, and then simply specify the s3a URL; for example:

```
call SYSCS_UTIL.IMPORT_DATA ('TPCH', 'REGION', null, 's3a://splice-benchmark-data/flat/TPCH/100/region', '|', null, null, null, null, 0, '/BAD', true, null);
```

To add your access and secret access keys to the `core-site.xml` file, define the `fs.s3a.awsAccessKeyId` and `fs.s3a.awsSecretAccessKey` properties in that file:

```
<property> <name>fs.s3a.awsAccessKeyId</name> <value>access key</value></property><property> <name>fs.s3a.awsSecretAccessKey</name> <value>secret key</value></property>
```

Input Data File Format

This section contains the following information about the format of the input data files that you're importing:

- » [Importing Compressed Files](#)
- » [Importing Multi-line Records](#)
- » [Importing Large Datasets in Groups of Files](#)

Importing Compressed Files

We recommend importing files that are either uncompressed, or have been compressed with `bz2` or `lz4` compression.

If you import files compressed with `gzip`, Splice Machine cannot distribute the contents of your file across your cluster nodes to take advantage of parallel processing, which means that import performance will suffer significantly with `gzip` files.

Importing Multi-line Records

If your data contains line feed characters like `CTRL-M`, you need to set the `oneLineRecords` parameter to `false`. Splice Machine will accommodate to the line feeds; however, the import will take longer because Splice Machine will not be able to break the file up and distribute it across the cluster.

To improve import performance, avoid including line feed characters in your data and set the `oneLineRecords` parameter to `true`.

Importing Large Datasets in Groups of Files

If you have a lot of data (100s of millions or billions of records), you may be tempted to create one massive file that contains all of your records and import that file; Splice Machine recommends against this; instead, we urge you to manage your data in smaller files. Specifically, we suggest that you split your data into files that are:

- » approximately 40 GB
- » have approximately 50 million records, depending on how wide your table is

If you have a lot of files, group them into multiple directories, and import each directory individually. For example, here is a structure our Customer Success engineers like to use:

- `/data/mytable1/group1`
- `/data/mytable1/group2`
- `/data/mytable1/group3`

Delimiters in Your Input Data

This section discusses the delimiters that you use in your input data, in these subsections:

- » [Using Special Characters for Delimiters](#)
- » [Column Delimiters](#)
- » [Character Delimiters](#)

Use Special Characters for Delimiters

One common gotcha we see with customer imports is when the data you're importing includes a special character that you've designated as a column or character delimiter. You'll end up with records in your bad record directory and can spend hours trying to determine the issue, only to discover that it's because the data includes a delimiter character. This can happen with columns that contain data such as product descriptions.

Column Delimiters

The standard column delimiter is a comma (,); however, we've all worked with string data that contains commas, and have figured out to use a different column delimiter. Some customers use the pipe (|) character, but frequently discover that it is also used in some descriptive data in the table they're importing.

In addition to using plain text characters, you can specify the following special characters as delimiters:

Special character	Display
\t	Tab
\f	Formfeed
\b	Backspace
\\\	Backslash
^a (or ^A)	Control-a

NOTE: If you are using a script file from the `splice>` command line, your script can contain the actual Control-a character as the value of this parameter.

We recommend using a control character like CTRL-A for your column delimiter. This is known as the SOH character, and is represented by 0x01 in hexadecimal. Unfortunately, there's no way to enter this character from the keyboard in the Splice Machine command line interface; instead, you need to [create a script file](#) and type the control character using a text editor like `vi` or `vim`:

- » Open your script file in `vi` or `vim`.
- » Enter into INSERT mode.
- » Type CTRL-V then CTRL-A for the value of the column delimiter parameter in your procedure call. Note that this typically echoes as ^A when you type it in `vi` or `vim`.

Character Delimiters

By default, the character delimiter is a double quote. This can produce the same kind of problems that we see with using a comma for the column delimiter: columns values that include embedded quotes or use the double quote as the symbol for inches. You can use escape characters to include the embedded quotes, but it's easier to use a special character for your delimiter.

We recommend using a control character like CTRL-A for your column delimiter. Unfortunately, there's no way to enter this character from the keyboard in the Splice Machine command line interface; instead, you need to [create a script file](#) and type the control character using a text editor like *vi* or *vim*:

- » Open your script file in vi or vim.
- » Enter into INSERT mode.
- » Type CTRL-V then CTRL-G for the value of the character delimiter parameter in your procedure call. Note that this typically echoes as ^G when you type it in vi or vim.

Time and Date Formats in Input Records

Perhaps the most common difficulty that customers have with importing their data is with date, time, and timestamp values.

Splice Machine adheres to the Java `SimpleDateFormat` syntax for all date, time, and timestamp values, `SimpleDateFormat` is described here:

[\[https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html\]\[1\]{: target=_blank}](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html)

Splice Machine's implementation of `SimpleDateFormat` is case-sensitive; this means, for example, that a lowercase `h` is used to represent an hour value between 0 and 12, whereas an uppercase `H` is used to represent an hour between 0 and 23.

All Values Must Use the Same Format

Splice Machine's Import procedures only allow you to specify one format each for the date, time, and timestamp columns in the table data you are importing. This means that, for example, every date in the table data must be in the same format.

All of the Date values in the file (or group of files) you are importing must use the same date format.

All of the Time values in the file (or group of files) you are importing must use the same time format.

All of the Timestamp values in the file (or group of files) you are importing must use the same timestamp format.

Additional Notes

A few additional notes:

- » The `Timestamp` data type has a range of 1678-01-01 to 2261-12-31. Some customers have used dummy timestamp values like 9999-01-01, which will fail because the value is out of range for a timestamp. Note that this is not an issue with `Date` values.
- » Splice Machine suggests that, if your data contains any date or timestamp values that are not in the format `yyyy-MM-dd HH:mm:ss`, you create a simple table that has just one or two columns and test importing the format. This is a simple

way to confirm that the imported data is what you expect.

- » Detailed information about each of these data types is found in our SQL Reference Manual:
 - » [Timestamp Data Type](#)
 - » [Date Data Type](#)
 - » [Time Data Type](#)

Importing and Updating Records

This section describes certain aspects of how records are imported and updated when you import data into your database, including these subsections:

- » [Inserting and Updating Column Values When Importing Data](#)
- » [Inserting and Updated Generated or Default Values](#)
- » [Handling Missing Values](#)

Inserting and Updating Column Values When Importing Data

This section summarizes what happens when you are importing, upserting, or merging records into a database table, based on:

- » Whether you are importing a new record or updating an existing record.
- » If the column is specified in your `insertColumnList` parameter.
- » If the table column is a generated value or has a default value.

The important difference in actions taken when importing data occurs when you are updating an existing record with the UPSERT or MERGE and your column list does not contain the name of a table column:

- » For newly inserted records, the default or auto-generated value is always inserted, as usual.
- » If you are updating an existing record in the table with UPSERT, the default auto-generated value in that record is overwritten with a new value.
- » If you are updating an existing record in the table with MERGE, the column value is not updated.

Importing a New Record Into a Database Table

The following table shows the actions taken when you are importing new records into a table in your database. These actions are the same for all three importation procedures (IMPORTing, UPSERTing, or MERGEing):

Column included in <code>importColumnList</code> ?	Table column conditions	Action Taken
YES	N/A	Import value inserted into table column if valid; if not valid, a bad record error is logged.

Column included in <code>importColumnList</code> ?	Table column conditions	Action Taken
NO	Has Default Value	Default value is inserted into table column.
	Is Generated Value	Generated value is inserted into table column.
	None	NULL is inserted into table column.

The table below shows what happens with default and generated column values when adding new records to a table using one of our import procedures; we use an example database table created with this statement:

```
CREATE TABLE myTable (
    colA INT,
    colB CHAR(12) DEFAULT 'myDefaultVal',
    colC INT);
```

insertColumnList	Values in import record	Values inserted into database	Notes
"colA,colB,colC"	1,,2	[1,NULL,2]	
"colA,colB,colC"	3,de,4	[3,de,4]	
"colA,colB,colC"	1,2,	Error: column B wrong type	
"colA,colB,colC"	1,DEFAULT,2	[1,"DEFAULT",2]	DEFAULT is imported as a literal value
Empty	1,,2	[1,myDefaultVal,2]	
Empty	3,de,4	[3,de,4]	
Empty	1,2,	Error: column B wrong type	
"colA,colC"	1,2	[1,myDefaultVal,2]	
"colA,colC"	3,4	[3,myDefaultVal,4]	

Note that the value 'DEFAULT' in the imported file **is not interpreted** to mean that the default value should be applied to that column; instead:

- » If the target column in your database has a string data type, such as CHAR or VARCHAR, the literal value "DEFAULT" is inserted into your database..
- » If the target column is not a string data type, an error will occur.

Importing Into a Table that Contains Generated or Default Values

When you export a table with generated columns to a file, the actual column values are exported, so importing that same file into a different database will accurately replicate the original table values.

If you are importing previously exported records into a table with a generated column, and you want to import some records with actual values and apply generated or default values to other records, you need to split your import file into two files and import each:

- » Import the file containing records with non-default values with the column name included in the `insertColumnList`.
- » Import the file containing records with default values with the column name excluded from the `insertColumnList`.

Updating a Table Record with UPSERT

The following table shows the action taken when you are using the `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` procedure to update an existing record in a database table:

Column included in <code>importColumnList</code> ?	Table column conditions	Action Taken
YES	N/A	Import value updated in table column if valid; if not valid, a bad record error is logged.
NO	Has Default Value	Table column is overwritten with default value.
	Is Generated Value	Table column is overwritten with newly generated value.
	None	Table column is overwritten with NULL value.

Updating a Table Record with MERGE

The following table shows the action taken when you are using the `SYSCS_UTIL.MERGE_DATA_FROM_FILE` procedure to update an existing record in a database table:

Column included in <code>importColumnList</code> ?	Table column conditions	Action Taken
YES	N/A	Import value updated in table column if valid; if not valid, a bad record error is logged.

Column included in <i>importColumnList</i> ?	Table column conditions	Action Taken
NO	N/A	Table column is not updated.

Importing CLOBs and BLOBS

If you are importing CLOBs, pay careful attention to tips [4](#) and [7](#). Be sure to use special characters for both your column and character delimiters. If your CLOB data can span multiple lines, be sure to set the `oneLineRecords` parameter to `false`.

At this time, the Splice Machine import procedures do not import work with columns of type BLOB. You can create a virtual table interface (VTI) that reads the BLOBS and inserts them into your database.

Scripting Your Imports

You can make import tasks much easier and convenient by creating *import scripts*. An import script is simply a call to one of the import procedures; once you've verified that it works, you can use and clone the script and run unattended imports.

An import script is simply a file in which you store `splice>` commands that you can execute with the `run` command. For example, here's an example of a text file named `myimports.sql` that we can use to import two csv files into our database:

```
call SYSCS_UTIL.IMPORT_DATA ('SPLICE','mytable1',null,'/data/mytable1/data.csv',null,null,null,null,null,0,'/BAD/mytable1',null,null);call SYSCS_UTIL.IMPORT_DATA ('SPLICE','mytable2',null,'/data/mytable2/data.csv',null,null,null,null,null,0,'/BAD/mytable2',null,null);
```

To run an import script, use the `splice> run` command; for example:

```
splice> run 'myimports.sql';
```

You can also start up the `splice>` command line interpreter with the name of a file to run; for example:

```
sqlshell.sh -f myimports.sql
```

In fact, you can script almost any sequence of Splice Machine commands in a file and run that script within the command line interpreter or when you start the interpreter.

See Also

- » [Importing Data: Tutorial Overview](#)
- » [Importing Data: Input Parameters](#)
- » [Importing Data: Error Handling](#)

- » Importing Data: Usage Examples
- » Importing Data: Bulk HFile Examples
- » Importing Data: Importing TPCH Data
- » [SYSCS_UTIL.IMPORT_DATA](#)
- » [SYSCS_UTIL.UPSERT_DATA_FROM_FILE](#)
- » [SYSCS_UTIL.MERGE_DATA_FROM_FILE](#)
- » [SYSCS_UTIL.BULK_IMPORT_HFILE](#)

Importing Data: Logging and Error Handling

This topic describes the logging and error handling features of Splice Machine data imports.

Logging

Each of these import procedures includes a logging facility:

- » [SYSCS_UTIL.IMPORT_DATA](#)
- » [SYSCS_UTIL.UPSERT_DATA_FROM_FILE](#)
- » [SYSCS_UTIL.MERGE_DATA_FROM_FILE](#)
- » [SYSCS_UTIL.BULK_IMPORT_HFILE](#)

Errors are logged to a file in the directory that you specify in the `badRecordDirectory` parameter when you call one of the procedures.

The `badRecordDirectory` parameter is a string that specifies the directory in which bad record information is logged. The default value is the directory in which the import files are found.

Splice Machine logs information to the `<import_file_name>.bad` file in this directory; for example, bad records in an input file named `foo.csv` would be logged to a file named `badRecordDirectory/foo.csv.bad`.

The `badRecordDirectory` directory must be writable by the `hbase` user, either by setting the user explicitly, or by opening up the permissions; for example:

```
sudo -su hdfs hadoop fs -chmod 777 /badRecordDirectory
```

NOTE: On a cluster, the `badRecordDirectory` directory **MUST be on S3, HDFS (or MapR-FS)**. If you're using our Database Service product, this directory must be on S3.

Stopping the Import Due to Too Many Errors

All of the import procedures also take a `badRecordsAllowed` or `maxBadRecords` parameter, the value of which determines how many erroneous input data record errors are allowed before the import is stopped. If this count of rejected records is reached, the import fails, and any successful record imports are rolled back.

These `badRecordsAllowed` values have special meaning:

- » If you specify `-1`, all record import failures are tolerated and logged.
- » If you specify `0`, the import will fail as soon as one bad record is detected.

Managing Logging When Importing Multiple Files

In addition to importing a single file, the `SYSICS_UTIL.IMPORT_DATA` and `SYSICS_UTIL.UPSERT_DATA_FROM_FILE` procedures can import all of the files in a directory.

When you are importing a large amount of data and have divided the files you are importing into groups, then it's a good idea to change the location of the bad record directory for each group; this will make debugging bad records a lot easier for you.

You can change the value of the `badRecordDirectory` to include your group name; for example, we typically use a strategy like the following:

Group Files Location	<code>badRecordDirectory</code> Parameter Value
/data/mytable1/group1	/BAD/mytable1/group1
/data/mytable1/group2	/BAD/mytable1/group2
/data/mytable1/group3	/BAD/mytable1/group3

You'll then be able to more easily discover where the problem record is located.

See Also

- » [Importing Data: Tutorial Overview](#)
- » [Importing Data: Input Parameters](#)
- » [Importing Data: Input Data Handling](#)
- » [Importing Data: Usage Examples](#)
- » [Importing Data: Bulk HFile Examples](#)
- » [Importing Data: Importing TPCH Data](#)
- » [`SYSICS_UTIL.IMPORT_DATA`](#)
- » [`SYSICS_UTIL.UPSERT_DATA_FROM_FILE`](#)
- » [`SYSICS_UTIL.MERGE_DATA_FROM_FILE`](#)
- » [`SYSICS_UTIL.BULK_IMPORT_HFILE`](#)

Importing Data: Examples of Using the Import, Upsert, and Merge Procedures

This topic provides several examples of importing data into Splice Machine using our *standard* import procedures (`IMPORT_DATA`, `UPSERT_DATA_FROM_FILE`, and `MERGE_DATA_FROM_FILE`):

- » [Example 1: Importing data into a table with fewer columns than in the file](#)
- » [Example 2: How Upsert and Merge handle missing columns differently](#)
- » [Example 3: Importing a subset of data from a file into a table](#)
- » [Example 4: Specifying a timestamp format for an entire table](#)
- » [Example 5: Importing strings with embedded special characters](#)
- » [Example 6: Using single quotes to delimit strings](#)

Import, Upsert, or Merge?

The Importing Data: Import Overview topic provides the information you need to decide which of our import procedures best meets your needs, including an easy-to-use decision tree.

To summarize, our three *standard* import procedures operate very similarly, with a few key differences:

- » `IMPORT_DATA` imports data into your database, creating a new record in your table for each record in the imported data.
- » `UPSERT_DATA_FROM_FILE` import data into your database, creating new records and *updating existing records* in the table. If a column is not specified in the input, `UPSERT_DATA_FROM_FILE` inserts the default value (or `NULL`, if no default) into that column in the imported record.
- » `MERGE_DATA_FROM_FILE` is identical to `UPSERT_DATA_FROM_FILE` except that it does not replace values in the table for unspecified columns when updating an existing record in the table.

A fourth option works differently:

- » `BULK_IMPORT_HFILE` creates temporary HFiles and imports from them, which improves import speed, but eliminates constraint checking and adds complexity. Examples of bulk HFile imports are found in the Importing Data: Bulk HFile Examples tutorial topic.

Example 1: Importing data into a table with fewer columns than in the file

If the table into which you're importing data has less columns than the data file that you're importing, how the "extra" data columns in the input data are handled depends on whether you specify an `insertColumnList`:

- » If you don't specify a specify an `insertColumnList` and your input file contains more columns than are in the table, then the extra columns at the end of each line in the input file are ignored. For example, if your table contains columns (`a`, `b`, `c`) and your file contains columns (`a`, `b`, `c`, `d`, `e`), then the data in your file's `d` and `e` columns will be ignored.

- » If you do specify an `insertColumnList` to `IMPORT_DATA` or `MERGE_DATA`, and the number of columns in your input file doesn't match the number in your table, then any other columns in your table will be replaced by the default value for the table column (or `NULL` if there is no default for the column). For example, if your table contains columns (a, b, c) and you only want to import columns (a, c), then the data in table's b column will be replaced with the default value (or `NULL`) for that column.

Here's an example that does not specify a column list. If you create a table with this statement:

```
CREATE TABLE playerTeams(ID int primary key, Team VARCHAR(32));
```

And your data file looks like this:

```
1,Cards,Molina,Catcher2,Giants,Posey,Catcher3,Royals,Perez,Catcher
```

When you import the file into `playerTeams`, only the first two columns are imported:

```
call SYSCS_UTIL.IMPORT_DATA('SPLICE','playerTeams',null, 'myData.csv',
    null, null, null, null, null, 0, 'importErrsDir', true, null);SELECT * FROM playe
rTeams ORDER by ID;ID      |TEAM
-----
1      |Cards2      |Giants
3      |Royals3 rows selected
```

How Missing Columns are Handled With an Insert Column List

In this example, we'll illustrate how the different data importation procedures modify columns in your table when you've specified an `insertColumnList` that is not 1-to-1 with the columns in your table.

The `SYSCS_UTIL.IMPORT_DATA` and `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` procedures handle this situation in the same way, assigning default values (or `NULL` if no default is defined) to any table column that is not being inserted or updated from the input data file. The `SYSCS_UTIL.MERGE_DATA_FROM_FILE` handles this differently: it does not overwrite generated values when updating records.

NOTE: This distinction is particularly important when loading record updates into a table with auto-generated column values that you do not want overwritten.

We'll create two sample tables, populate each with the same data, and load the same input file data into each to illustrate the differences between how the `Upsert` and `Merge` procedures.

```

CREATE SCHEMA test;
SET SCHEMA test;
CREATE TABLE testUpsert (
    a1 INT,
    b1 INT,
    c1 INT GENERATED BY DEFAULT AS IDENTITY(start with 1, increment by 1),
    d1 INT DEFAULT 999,
    PRIMARY KEY (a1)
);

CREATE TABLE testMerge (
    a1 INT,
    b1 INT,
    c1 INT GENERATED BY DEFAULT AS IDENTITY(start with 1, increment by 1),
    d1 INT DEFAULT 999,
    PRIMARY KEY (a1)
);

INSERT INTO testUpsert(a1,b1) VALUES (1,1), (2,2), (3,3), (6,6);
splice> select * from testUpsert;
A1      |B1       |C1       |D1
-----
1      |1        |1        |999
2      |2        |2        |999
3      |3        |3        |999
6      |6        |4        |999

4 rows selected

INSERT INTO testMerge (a1,b1) VALUES (1,1), (2,2), (3,3), (6,6);
splice> select * from testMerge;
A1      |B1       |C1       |D1
-----
1      |1        |1        |999
2      |2        |2        |999
3      |3        |3        |999
6      |6        |4        |999

4 rows selected

```

Note that column `c1` contains auto-generated values, and that column `d1` has the default value 999.

Here's the data that we're going to import from file `ttest.csv`:

0	0
1	2
2	4
3	6
4	8

Now, let's call UPSERT_DATA_FROM_FILE and MERGE_DATA_FROM_FILE and see how the results differ:

```

CALL SYSCS_UTIL.UPSERT_DATA_FROM_FILE('TEST','testUpsert','a1,b1','/Users/garyh/Documents/ttest.csv','|',null,null,null,null,0,'/var/tmp/bad/',false,null);
rowsImported      | failedRows      | files      | dataSize      | failedLog
-----
5                | 0              | 1          | 20          |
                                         | NONE        |

splice> SELECT * FROM testUpsert;
A1      | B1      | C1      | D1
-----
0      | 0      | 10001   | 999
1      | 2      | 10002   | 999
2      | 4      | 10003   | 999
3      | 6      | 10004   | 999
4      | 8      | 10005   | 999
6      | 6      | 4        | 999

6 rows selected

CALL SYSCS_UTIL.MERGE_DATA_FROM_FILE('TEST','testMerge','a1,b1','/Users/garyh/Documents/ttest.csv','|',null,null,null,null,0,'/var/tmp/bad/',false,null);
rowsUpdated      | rowsInserted    | failedRows    | files      | dataSize
e              | failedLog
-----
3                | 2              | 0            | 1          | 2
0                | NONE           |              |             |
                                         |              |             |

splice> select * from testMerge;
A1      | B1      | C1      | D1
-----
0      | 0      | 10001   | 999
1      | 2      | 1        | 999
2      | 4      | 2        | 999
3      | 6      | 3        | 999
4      | 8      | 10002   | 999
6      | 6      | 4        | 999

6 rows selected

```

You'll notice that:

- » The generated column (c1) is not included in the insertColumnList parameter in these calls.
- » The results are identical except for the values in the generated column.
- » The generated values in c1 are not updated in existing records when merging data, but are updated when upserting data.

Example 3: Importing a subset of data from a file into a table

This example uses the same table and import file as does the previous example, and it produces the same results. The difference between these two examples is that this one explicitly imports only the first two columns (which are named ID and TEAM) of the file and uses the IMPORT_DATA procedure:

```
call SYSCS_UTIL.IMPORT_DATA('SPLICE','playerTeams', 'ID, TEAM', 'myData.csv',
  null, null, null, null, null, 0, 'importErrsDir', true, null);SELECT * FROM playerT
eams ORDER by ID;ID    |TEAM
-----
1  |Cards
2  |Giants
3  |Royal
s3 rows selected
```

Example 4: Specifying a timestamp format for an entire table

This examples demonstrates how you can use a single timestamp format for the entire table by explicitly specifying a single timeStampFormat. Here's the data:

```
Mike,2013-04-21 09:21:24.98-05
Mike,2013-04-21 09:15:32.78-04
Mike,2013-03-23 09:45:00.68-05
```

You can then import the data with the following call:

```
call SYSCS_UTIL.IMPORT_DATA('app','tabx','c1,c2',
  '/path/to/ts3.csv',
  ',',
  'yyyy-MM-dd HH:mm:ss.SSZ',
  null, null, 0, null, true, null);
```



Note that the time shown in the imported table depends on the timezone setting in the server timestamp. In other words, given the same csv file, if imported on different servers with timestamps set to different time zones, the value in the table shown will be different. Additionally, daylight savings time may account for a 1-hour difference if timezone is specified.

Example 5: Importing strings with embedded special characters

This example imports a csv file that includes newline (Ctrl-M) characters in some of the input strings. We use the default double-quote character as our character delimiter to import data such as the following:

```
1,This field is one line,Able
2,"This field has two lines
This is the second line of the field",Baker
3,This field is also just one line,Charlie
```

We then use the following call to import the data:

```
SYSCS_UTIL.IMPORT_DATA('SPLICE', 'MYTABLE', null, 'data.csv' , '\t', null, null, null, null, 0, 'importErrsDir', false, null);
```

We can also explicitly specify double quotes (or any other character) as our delimiter character for strings:

```
SYSCS_UTIL.IMPORT_DATA('SPLICE', 'MYTABLE', null, 'data.csv', '\t', '"', null, null, null, null, 0, 'importErrsDir', false, null);
```

Example 6: Using single quotes to delimit strings

This example performs the same import as the previous example, simply substituting single quotes for double quotes as the character delimiter in the input:

```
1,This field is one line,Able
2,'This field has two lines
This is the second line of the field',Baker
3,This field is also just one line,Charlie
```

Note that you must escape single quotes in SQL, which means that you actually define the character delimiter parameter with four single quotes, as shown here:

```
SYSCS_UTIL.IMPORT_DATA('SPLICE', 'MYTABLE', null, 'data.csv', '\t', "''", null, null, null, null, 0, 'importErrsDir', false, null);
```

See Also

- » [Importing Data: Tutorial Overview](#)
- » [Importing Data: Input Parameters](#)
- » [Importing Data: Input Data Handling](#)
- » [Importing Data: Error Handling](#)
- » [Importing Data: Bulk HFile Examples](#)
- » [Importing Data: Importing TPCH Data](#)
- » [**SYSCS_UTIL.IMPORT_DATA**](#)
- » [**SYSCS_UTIL.UPSERT_DATA_FROM_FILE**](#)
- » [**SYSCS_UTIL.MERGE_DATA_FROM_FILE**](#)
- » [**SYSCS_UTIL.BULK_IMPORT_HFILE**](#)

Importing Data With the Bulk HFile Import Procedure

This tutorial describes how to import data using HFiles into your Splice Machine database with the `SYSCS_UTIL.BULK_IMPORT_HFILE` system procedure. This topic includes these sections:

- » [How Importing Your Data as HFiles Works](#) presents an overview of using the HFile import functions.
- » [Configuration Settings](#) describes any configuration settings that you may need to modify when using the `SYSCS_UTIL.BULK_IMPORT_HFILE` procedure to import data into your database.
- » [Importing Data From the Cloud](#) links to our instructions for configuring Splice Machine access to your data in the cloud.
- » [Manually Computing Table Splits](#) outlines the steps you use to manually compute table splits, if you prefer to not have that handled automatically.
- » [Examples of Using SYSCS_UTIL.BULK_IMPORT_HFILE](#) walks through using this procedure both with automatic table splits and with two different methods of manually computing table splits.

Our Importing Data: Usage Examples topic walks you through using our standard import procedures (`SYSCS_UTIL.IMPORT_DATA`, `SYSCS_UTIL.SYSCS_UPSERT_DATA_FROM_FILE`, and `SYSCS_UTIL.SYSCS_MERGE_DATA_FROM_FILE`), which are simpler to use, though their performance is slightly lower than importing HFiles.



Bulk importing HFiles boosts import performance; however, constraint checking is not applied to the imported data. If you need constraint checking, use one of our standard import procedures.

How Importing Your Data as HFiles Works

Our HFile data import procedure leverages HBase bulk loading, which allows it to import your data at a faster rate; however, using this procedure instead of our standard `SYSCS_UTIL.IMPORT_DATA` procedure means that *constraint checks are not performing during data importation*.

You import a table as HFiles using our `SYSCS_UTIL.BULK_IMPORT_HFILE` procedure, which temporarily converts the table file that you're importing into HFiles, imports those directly into your database, and then removes the temporary HFiles.

Before it generate HFiles, `SYSCS_UTIL.BULK_IMPORT_HFILE` must determine how to split the data into multiple regions by looking at the primary keys and figuring out which values will yield relatively evenly-sized splits; the objective is to compute splits such that roughly the same number of table rows will end up in each split.

You have two choices for determining the table splits:

- » You can have `SYSCS_UTIL.BULK_IMPORT_HFILE` scan and analyze your table to determine the best splits automatically by calling `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter set to `true`. We walk you through using this approach in the first example below, [Example 1: Bulk HFile Import with Automatic Table Splitting](#)

- » You can compute the splits yourself and then call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter set to `false`. Computing the splits requires these steps, which are described in the next section, [Manually Computing Table Splits](#).

1. Determine which values make sense for splitting your data into multiple regions. This means looking at the primary keys for the table and figuring out which values will yield relatively evenly-sized (in number of rows) splits.
2. Call our system procedures to compute the HBase-encoded keys and set up the splits inside your Splice Machine database.
3. Call the `SYSCS_UTIL.BULK_IMPORT_HFILE` procedure with the `skipSampling` parameter to `false` to perform the import.

Configuration Settings

Due to how Yarn manages memory, you need to modify your YARN configuration when bulk-importing large datasets. Make these two changes in your Yarn configuration:

```
yarn.nodemanager.pmem-check-enabled=false
yarn.nodemanager.vmem-check-enabled=false
```

Importing Data From the Cloud

If you are importing data that is stored in an S3 bucket on AWS, you need to specify the data location in an `s3a` URL that includes access key information. Our [Configuring an S3 Bucket for Splice Machine Access](#) walks you through using your AWS dashboard to generate and apply the necessary credentials.

Manually Computing Table Splits

If you're computing splits for your import (and calling the `SYSCS_UTIL.BULK_IMPORT_HFILE` procedure with `skipSampling` parameter to `false`), you need to select one of these two methods of computing the table splits:

- » You can call `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX` to compute the splits; the [Example 2](#) example walks you through this.

-or-

- » You can call `SYSCS_UTIL.COMPUTE_SPLIT_KEY` to generate a keys file, and then call `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS` to set up the splits in your database; the [Example 3](#) example walks you through this.

In either case, after computing the splits, you call `SYSCS_UTIL.BULK_IMPORT_HFILE` to split your input file into HFiles, import your data, and then remove the temporary HFiles.

Here's a quick summary of how you can compute your table splits:

1. Create a directory on HDFS for the import; for example:

```
sudo -su hdfs hadoop fs -mkdir hdfs://tmp/test_hfile_import
```

2. Determine primary key values that can horizontally split the table into roughly equal sized partitions.

Ideally, each partition should be about 1/2 the size of your `hbase.hregion.max.filesize` setting, which leaves room for the region to grow after your data is imported.

The size of each partition **must be less than** the value of `hbase.hregion.max.filesize`.

3. Store those keys in a CSV file.
4. Compute the split keys and then split the table.
5. Repeat steps 1, 2, and 3 to split the indexes on your table.
6. Call the `SYSCS_UTIL.BULK_IMPORT_HFILE` procedure to split the input data file into HFiles and import the HFiles into your Splice Machine database. The HFiles are automatically deleted after being imported.

You'll find detailed descriptions of these steps in these two examples:

- » [Example 2: Using `SPLIT_TABLE_OR_INDEX` to Compute Table Splits](#)
- » [Example 3: Using `SYSCS_UTIL.COMPUTE_SPLIT_KEY` and `SPLIT_TABLE_OR_INDEX_AT_POINTS` to Compute Table Splits.](#)

Examples of Using `SYSCS_UTIL.BULK_IMPORT_HFILE`

This section contains example walkthroughs of using the `SYSCS_UTIL.BULK_IMPORT_HFILE` system procedure in three different ways:

- » [Example 1](#) uses the automatic table splitting built into `SYSCS_UTIL.BULK_IMPORT_HFILE` to import a table into your Splice Machine database.
- » [Example 2](#) uses the `SYSCS_UTIL.COMPUTE_SPLIT_TABLE_OR_INDEX` system procedure to calculate the table splits before calling `SYSCS_UTIL.BULK_IMPORT_HFILE` to import a table into your Splice Machine database.
- » [Example 3](#) uses the `SYSCS_UTIL.COMPUTE_SPLIT_KEY` and `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS` system procedures to calculate the table splits before calling `SYSCS_UTIL.BULK_IMPORT_HFILE` to import a table into your Splice Machine database.

Example 1: Bulk HFile Import with Automatic Table Splitting

This example details the steps used to import data in HFile format using the Splice Machine SYSCS_UTIL.BULK_IMPORT_HFILE system procedure with automatic splitting.

Follow these steps:

- 1. Create a directory on HDFS for the import; for example:**

```
sudo -su hdfs hadoop fs -mkdir hdfs://tmp/test_hfile_import
```

Make sure that the directory you create has permissions set to allow Splice Machine to write your csv and Hfiles there.

- 2. Create table and index:**

```
CREATE TABLE TPCH.LINEITEM (
    L_ORDERKEY BIGINT NOT NULL,
    L_PARTKEY INTEGER NOT NULL,
    L_SUPPKEY INTEGER NOT NULL,
    L_LINENUMBER INTEGER NOT NULL,
    L_QUANTITY DECIMAL(15,2),
    L_EXTENDEDPRICE DECIMAL(15,2),
    L_DISCOUNT DECIMAL(15,2),
    L_TAX DECIMAL(15,2),
    L_RETURNFLAG VARCHAR(1),
    L_LINESSTATUS VARCHAR(1),
    L_SHIPDATE DATE,
    L_COMMITDATE DATE,
    L_RECEIPTDATE DATE,
    L_SHIPINSTRUCT VARCHAR(25),
    L_SHIPMODE VARCHAR(10),
    L_COMMENT VARCHAR(44),
    PRIMARY KEY(L_ORDERKEY,L_LINENUMBER)
);

CREATE INDEX L_SHIPDATE_IDX on TPCH.LINEITEM(
    L_SHIPDATE,
    L_PARTKEY,
    L_EXTENDEDPRICE,
    L_DISCOUNT
);
```

- 3. Import the HFiles Into Your Database**

Once you have split your table and indexes, call this procedure to generate and import the HFiles into your Splice Machine database:

```
call SYSCS_UTIL.BULK_IMPORT_HFILE('TPCH', 'LINEITEM', null,
    '/TPCH/1/lineitem', '||', null, null, null, null, -1,
    '/BAD', true, null, 'hdfs:///tmp/test_hfile_import/', false);
```

The generated HFiles are automatically deleted after being imported.

Example 2: Using **SPLIT_TABLE_OR_INDEX** to Compute Table Splits

The example in this section details the steps used to import data in HFile format using the Splice Machine `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX` and `SYSCS_UTIL.BULK_IMPORT_HFILE` system procedures.

Follow these steps:

1. Create a directory on HDFS for the import; for example:

```
sudo -su hadoop fs -mkdir hdfs:///tmp/test_hfile_import
```

Make sure that the directory you create has permissions set to allow Splice Machine to write your csv and Hfiles there.

2. Create table and index:

```

CREATE TABLE TPCH.LINEITEM (
    L_ORDERKEY BIGINT NOT NULL,
    L_PARTKEY INTEGER NOT NULL,
    L_SUPPKEY INTEGER NOT NULL,
    L_LINENUMBER INTEGER NOT NULL,
    L_QUANTITY DECIMAL(15,2),
    L_EXTENDEDPRICE DECIMAL(15,2),
    L_DISCOUNT DECIMAL(15,2),
    L_TAX DECIMAL(15,2),
    L_RETURNFLAG VARCHAR(1),
    L_LINESSTATUS VARCHAR(1),
    L_SHIPDATE DATE,
    L_COMMITDATE DATE,
    L_RECEIPTDATE DATE,
    L_SHIPINSTRUCT VARCHAR(25),
    L_SHIPMODE VARCHAR(10),
    L_COMMENT VARCHAR(44),
    PRIMARY KEY(L_ORDERKEY,L_LINENUMBER)
);

CREATE INDEX L_SHIPDATE_IDX on TPCH.LINEITEM(
    L_SHIPDATE,
    L_PARTKEY,
    L_EXTENDEDPRICE,
    L_DISCOUNT
);

```

3. Compute the split row keys for your table and set up the split in your database:

- a. Find primary key values that can horizontally split the table into roughly equal sized partitions.

For this example, we provide 3 keys in a file named `lineitemKey.csv`. Note that each of our three keys includes a second column that is `null`:

1500000
3000000
4500000

For every N lines of split data you specify, you'll end up with N+1 regions; for example, the above 3 splits will produce these 4 regions:

0 -> 1500000
1500000 -> 3000000
3000000 -> 4500000
4500000 -> (last possible key)

- b. Specify the column names in the csv file in the `columnList` parameter; in our example, the primary key columns are:

```
'L_ORDERKEY,L_LINENUMBER'
```

- c. Invoke `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX` to compute hbase split row keys and set up the splits

```
call SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX('TPCH',
    'LINEITEM', null, 'L_ORDERKEY,L_LINENUMBER',
    'hdfs:///tmp/test_hfile_import/lineitemKey.csv',
    '|', null, null, null,
    null, -1, '/BAD', true, null);
```

4. Compute the split keys for your index:

- a. Find index values that can horizontally split the table into roughly equal sized partitions.
- b. For this example, we provide 2 index values in a file named `shipDateIndex.csv`. Note that each of our keys includes `null` column values:

```
1994-01-01|||
1996-01-01|||
```

- c. Specify the column names in the csv file in the `columnList` parameter; in our example, the index columns are:

```
'L_SHIPDATE,L_PARTKEY,L_EXTENDEDPRICE,L_DISCOUNT'
```

- d. Invoke `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX` to compute hbase split row keys and set up the index splits

```
call SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX('TPCH',
    'LINEITEM', 'L_SHIPDATE_IDX',
    'L_SHIPDATE,L_PARTKEY,L_EXTENDEDPRICE,L_DISCOUNT',
    'hdfs:///tmp/test_hfile_import/shipDateIndex.csv',
    '|', null, null,
    null, null, -1, '/BAD', true, null);
```

5. Import the HFiles Into Your Database

Once you have split your table and indexes, call this procedure to generate and import the HFiles into your Splice Machine database:

```
call SYSCS_UTIL.BULK_IMPORT_HFILE('TPCH', 'LINEITEM', null,
    '/TPCH/1/lineitem', '|', null, null, null, null,
    -1, '/BAD', true, null,
    'hdfs:///tmp/test_hfile_import/', true);
```

The generated HFiles are automatically deleted after being imported.

Example 3: Using `SYSCS_UTIL.COMPUTE_SPLIT_KEY` and `SPLIT_TABLE_OR_INDEX_AT_POINTS` to Compute Table Splits

The example in this section details the steps used to import data in HFile format using the Splice Machine `SYSCS_UTIL.COMPUTE_SPLIT_KEY`, `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS`, and `SYSCS_UTIL.BULK_IMPORT_HFILE` system procedures.

Follow these steps:

1. Create a directory on HDFS for the import; for example:

```
sudo -su hdfs hadoop fs -mkdir hdfs:///tmp/test_hfile_import
```

Make sure that the directory you create has permissions set to allow Splice Machine to write your csv and Hfiles there.

2. Create table and index:

```
CREATE TABLE TPCH.LINEITEM (
    L_ORDERKEY BIGINT NOT NULL,
    L_PARTKEY INTEGER NOT NULL,
    L_SUPPKEY INTEGER NOT NULL,
    L_LINENUMBER INTEGER NOT NULL,
    L_QUANTITY DECIMAL(15,2),
    L_EXTENDEDPRICE DECIMAL(15,2),
    L_DISCOUNT DECIMAL(15,2),
    L_TAX DECIMAL(15,2),
    L_RETURNFLAG VARCHAR(1),
    L_LINESSTATUS VARCHAR(1),
    L_SHIPDATE DATE,
    L_COMMITDATE DATE,
    L_RECEIPTDATE DATE,
    L_SHIPINSTRUCT VARCHAR(25),
    L_SHIPMODE VARCHAR(10),
    L_COMMENT VARCHAR(44),
    PRIMARY KEY(L_ORDERKEY,L_LINENUMBER)
);
```

3. Compute the split row keys for the table:

- a. Find primary key values that can horizontally split the table into roughly equal sized partitions.

For this example, we provide 3 keys in a file named `lineitemKey.csv`. Note that each of our three keys includes a second column that is `null`:

```
1500000 |
3000000 |
4500000 |
```

For every N lines of split data you specify, you'll end up with N+1 regions; for example, the above 3 splits will produce these 4 regions:

```
0 -> 1500000
1500000 -> 3000000
3000000 -> 4500000
4500000 -> (last possible key)
```

- b. Specify the column names in the csv file in the `columnList` parameter; in our example, the primary key columns are:

```
'L_ORDERKEY,L_LINENUMBER'
```

- c. Invoke `SYSCS_UTIL.COMPUTE_SPLIT_KEY` to compute hbase split row keys and write them to a file:

```
call SYSCS_UTIL.COMPUTE_SPLIT_KEY('TPCH', 'LINEITEM',
    null, 'L_ORDERKEY,L_LINENUMBER',
    'hdfs:///tmp/test_hfile_import/lineitemKey.csv',
    '|', null, null, null,
    null, -1, '/BAD', true, null, 'hdfs:///tmp/test_hfile_import/');
```

4. Set up the table splits in your database:

- a. Use `SHOW TABLES` to discover the conglomerate ID for the `TPCH.LINEITEM` table, which for this example is 1536. This means that the split keys file for this table is in the `hdfs:///tmp/test_hfile_import/1536` directory. You'll see values like these:

```
\xE4\x16\xE3`\xE4-\xC6\xC0\xE4D\xAA
```

- b. Now use those values in a call to our system procedure to split the table inside the database:

```
call SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS('TPCH', 'LINEITEM',
    null, '\xE4\x16\xE3`\xE4-\xC6\xC0\xE4D\xAA');
```

5. Compute the split keys for your index:

- Find index values that can horizontally split the table into roughly equal sized partitions.
- For this example, we provide 2 index values in a file named `shipDateIndex.csv`. Note that each of our keys includes `null` column values:

```
1994-01-01|||
1996-01-01|||
```

- Specify the column names in the csv file in the `columnList` parameter; in our example, the index columns are:

```
'L_SHIPDATE,L_PARTKEY,L_EXTENDEDPRICE,L_DISCOUNT'
```

- Invoke `SYSCS_UTIL.COMPUTE_SPLIT_KEY` to compute hbase split row keys and write them to a file:

```
call SYSCS_UTIL.COMPUTE_SPLIT_KEY('TPCH', 'LINEITEM', 'L_SHIPDATE_ID
X',
'L_SHIPDATE,L_PARTKEY,L_EXTENDEDPRICE,L_DISCOUNT',
'hdfs:///tmp/test_hfile_import/shipDateIndex.csv',
'|', null, null, null, null, -1, '/BAD', true, null,
'hdfs:///tmp/test_hfile_import/');
```

6. Set up the indexes in your database:

- Copy the row key values from the output file:

```
\xEC\xB0Y9\xBC\x00\x00\x00\x00\x00\x80
\xEC\xBF\x08\x9C\x14\x00\x00\x00\x00\x80
```

- Now call our system procedure to split the index:

```
call SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS(
    'TPCH', 'LINEITEM', 'L_SHIPDATE_IDX',
    '\xEC\xB0Y9\xBC\x00\x00\x00\x00\x80,
    \xEC\xBF\x08\x9C\x14\x00\x00\x00\x00\x80');
```

7. Import the HFiles Into Your Database

Once you have split your table and indexes, call this procedure to generate and import the HFiles into your Splice Machine database:

```
call SYSCS_UTIL.BULK_IMPORT_HFILE('TPCH', 'LINEITEM', null,
    '/TPCH/1/lineitem', '|', null, null, null, null, -1,
    '/BAD', true, null,
    'hdfs:///tmp/test_hfile_import/', true);
```

The generated HFiles are automatically deleted after being imported.

See Also

- » Importing Data: Tutorial Overview
- » Importing Data: Input Parameters
- » Importing Data: Input Data Handling
- » Importing Data: Error Handling
- » Importing Data: Usage Examples
- » Importing Data: Importing TPCH Data
- » [SYSCS_UTIL.IMPORT_DATA](#)
- » [SYSCS_UTIL.UPSERT_DATA_FROM_FILE](#)
- » [SYSCS_UTIL.MERGE_DATA_FROM_FILE](#)
- » [SYSCS_UTIL.BULK_IMPORT_HFILE](#)

Importing TPCH Data Into Your Database

This topic walks you through importing the TPCH sample data into your Splice Machine database and then querying that data, in these sections:

- » [Import TPCH Data](#) walks you through importing TPCH data from our AWS bucket into your Splice Machine database.
- » [Importing Your Own Data](#) links to our tutorial that helps you to import your own data.
- » [The TPCH Queries](#) includes the SQL source for each of the TPCH queries, so you can quickly run any of them against your newly imported data.

Import TPCH Data

You can use the following steps to import TPCH data into your new Splice Machine database:

1. Create the schema and tables

You can copy/paste the following SQL statements to create the schema and tables for importing the sample data:

```

CREATE SCHEMA TPCH;

CREATE TABLE TPCH.LINEITEM (
    L_ORDERKEY BIGINT NOT NULL,
    L_PARTKEY INTEGER NOT NULL,
    L_SUPPKEY INTEGER NOT NULL,
    L_LINENUMBER INTEGER NOT NULL,
    L_QUANTITY DECIMAL(15,2),
    L_EXTENDEDPRICE DECIMAL(15,2),
    L_DISCOUNT DECIMAL(15,2),
    L_TAX DECIMAL(15,2),
    L_RETURNFLAG VARCHAR(1),
    L_LINESSTATUS VARCHAR(1),
    L_SHIPDATE DATE,
    L_COMMITDATE DATE,
    L_RECEIPTDATE DATE,
    L_SHIPINSTRUCT VARCHAR(25),
    L_SHIPMODE VARCHAR(10),
    L_COMMENT VARCHAR(44),
    PRIMARY KEY(L_ORDERKEY,L_LINENUMBER)
);

CREATE TABLE TPCH.ORDERS (
    O_ORDERKEY BIGINT NOT NULL PRIMARY KEY,
    O_CUSTKEY INTEGER,
    O_ORDERSTATUS VARCHAR(1),
    O_TOTALPRICE DECIMAL(15,2),
    O_ORDERDATE DATE,
    O_ORDERPRIORITY VARCHAR(15),
    O_CLERK VARCHAR(15),
    O_SHIPPRIORITY INTEGER ,
    O_COMMENT VARCHAR(79)
);

CREATE TABLE TPCH.CUSTOMER (
    C_CUSTKEY INTEGER NOT NULL PRIMARY KEY,
    C_NAME VARCHAR(25),
    C_ADDRESS VARCHAR(40),
    C_NATIONKEY INTEGER NOT NULL,
    C_PHONE VARCHAR(15),
    C_ACCTBAL DECIMAL(15,2),
    C_MKTSEGMENT VARCHAR(10),
    C_COMMENT VARCHAR(117)
);

CREATE TABLE TPCH.PARTSUPP (
    PS_PARTKEY INTEGER NOT NULL ,
    PS_SUPPKEY INTEGER NOT NULL ,

```

```

PS_AVAILQTY INTEGER,
PS_SUPPLYCOST DECIMAL(15,2),
PS_COMMENT VARCHAR(199),
PRIMARY KEY(PS_PARTKEY,PS_SUPPKEY)
);

CREATE TABLE TPCH.SUPPLIER (
S_SUPPKEY INTEGER NOT NULL PRIMARY KEY,
S_NAME VARCHAR(25) ,
S_ADDRESS VARCHAR(40) ,
S_NATIONKEY INTEGER ,
S_PHONE VARCHAR(15) ,
S_ACCTBAL DECIMAL(15,2),
S_COMMENT VARCHAR(101)
);

CREATE TABLE TPCH.PART (
P_PARTKEY INTEGER NOT NULL PRIMARY KEY,
P_NAME VARCHAR(55) ,
P_MFGR VARCHAR(25) ,
P_BRAND VARCHAR(10) ,
P_TYPE VARCHAR(25) ,
P_SIZE INTEGER ,
P_CONTAINER VARCHAR(10) ,
P_RETAILPRICE DECIMAL(15,2),
P_COMMENT VARCHAR(23)
);

CREATE TABLE TPCH.REGION (
R_REGIONKEY INTEGER NOT NULL PRIMARY KEY,
R_NAME VARCHAR(25),
R_COMMENT VARCHAR(152)
);

CREATE TABLE TPCH.NATION (
N_NATIONKEY INTEGER NOT NULL,
N_NAME VARCHAR(25),
N_REGIONKEY INTEGER NOT NULL,
N_COMMENT VARCHAR(152),
PRIMARY KEY (N_NATIONKEY)
);

```

2. Import data

We've put a copy of the TPCH data in an AWS S3 bucket for convenient retrieval. You can copy/paste the following `SYSICS_UTIL.IMPORT_DATA` statements to quickly pull that data into your database:

```

call SYSCS_UTIL.IMPORT_DATA ('TPCH', 'LINEITEM', null, 's3a:/splice-bench
mark-data/flat/TPCH/1/lineitem', ' | ', null, null, null, null, 0, '/tmp/BA
D', true, null);

call SYSCS_UTIL.IMPORT_DATA ('TPCH', 'ORDERS', null, 's3a:/splice-bench
mark-data/flat/TPCH/1/orders', ' | ', null, null, null, null, 0, '/tmp/BA
D', true, null);

call SYSCS_UTIL.IMPORT_DATA ('TPCH', 'CUSTOMER', null, 's3a:/splice-bench
mark-data/flat/TPCH/1/customer', ' | ', null, null, null, null, 0, '/tmp/BA
D', true, null);

call SYSCS_UTIL.IMPORT_DATA ('TPCH', 'PARTSUPP', null, 's3a:/splice-bench
mark-data/flat/TPCH/1/partsupp', ' | ', null, null, null, null, 0, '/tmp/BA
D', true, null);

call SYSCS_UTIL.IMPORT_DATA ('TPCH', 'SUPPLIER', null, 's3a:/splice-bench
mark-data/flat/TPCH/1/supplier', ' | ', null, null, null, null, 0, '/tmp/BA
D', true, null);

call SYSCS_UTIL.IMPORT_DATA ('TPCH', 'PART', null, 's3a:/splice-bench
mark-data/flat/TPCH/1/part', ' | ', null, null, null, null, 0, '/tmp/BA
D', true, null);

call SYSCS_UTIL.IMPORT_DATA ('TPCH', 'REGION', null, 's3a:/splice-bench
mark-data/flat/TPCH/1/region', ' | ', null, null, null, null, 0, '/tmp/BA
D', true, null);

call SYSCS_UTIL.IMPORT_DATA ('TPCH', 'NATION', null, 's3a:/splice-bench
mark-data/flat/TPCH/1/nation', ' | ', null, null, null, null, 0, '/tmp/BA
D', true, null);

```

3. Run a query

You can now copy/paste *TPCH Query 01* against the imported data to verify that all's well:

```
-- QUERY 01
select
    l_returnflag,
    l_linenstatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    TPCH.lineitem
where
    l_shipdate = date({fn TIMESTAMPADD(SQL_TSI_DAY, -90, cast('1998-12-01 00:00:00' as timestamp)))})
group by
    l_returnflag,
    l_linenstatus
order by
    l_returnflag,
    l_linenstatus
-- END OF QUERY
```

We've also included the SQL for most of the other [TPCH queries](#) in this topic, should you want to try others.

Importing Your Own Data

You can follow similar steps to import your own data. Setting up your import requires some precision; we encourage you to look through our Importing Your Data Tutorial for guidance and tips to make that process go smoothly.

The TPCH Queries

Here are a number of additional queries you might want to run against the TPCH data:

Query01

```
-- QUERY 01
select
    l_returnflag,
    l_linenstatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    TPCH.lineitem
where
    l_shipdate = date({fn TIMESTAMPADD(SQL_TSI_DAY, -90, cast('199
8-12-01 00:00:00' as timestamp))})
group by
    l_returnflag,
    l_linenstatus
order by
    l_returnflag,
    l_linenstatus
-- END OF QUERY
```

Query02

```
-- QUERY 02
select
    s_acctbal,
    s_name,
    n_name,
    p_partkey,
    p_mfgr,
    s_address,
    s_phone,
    s_comment
from
    TPCH.part,
    TPCH.supplier,
    TPCH.partsupp,
    TPCH.nation,
    TPCH.region
where
    p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_size = 15
    and p_type like '%BRASS'
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'EUROPE'
    and ps_supplycost = (
        select
            min(ps_supplycost)
        from
            TPCH.partsupp,
            TPCH.supplier,
            TPCH.nation,
            TPCH.region
        where
            p_partkey = ps_partkey
            and s_suppkey = ps_suppkey
            and s_nationkey = n_nationkey
            and n_regionkey = r_regionkey
            and r_name = 'EUROPE'
    )
order by
    s_acctbal desc,
    n_name,
    s_name,
    p_partkey
{limit 100}
```

Query03

```
-- QUERY 03
select
    l_orderkey,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    o_orderdate,
    o_shipppriority
from
    TPCH.customer,
    TPCH.orders,
    TPCH.lineitem
where
    c_mktsegment = 'BUILDING'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate  date('1995-03-15')
    and l_shipdate   date('1995-03-15')
group by
    l_orderkey,
    o_orderdate,
    o_shipppriority
order by
    revenue desc,
    o_orderdate
{limit 10}
-- END OF QUERY
```

Query04

```
-- QUERY 04
select
    o_orderpriority,
    count(*) as order_count
from
    TPCH.orders
where
    o_orderdate >= date('1993-07-01')
    and o_orderdate < add_months('1993-07-01',3)
    and exists (
        select
            *
        from
            TPCH.lineitem
        where
            l_orderkey = o_orderkey
            and l_commitdate < l_receiptdate
    )
group by
    o_orderpriority
order by
    o_orderpriority
-- END OF QUERY
```

Query05

```
-- QUERY 05
select
    n_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue
from
    TPCH.customer,
    TPCH.orders,
    TPCH.lineitem,
    TPCH.supplier,
    TPCH.nation,
    TPCH.region
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and l_suppkey = s_suppkey
    and c_nationkey = s_nationkey
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'ASIA'
    and o_orderdate >= date('1994-01-01')
    and o_orderdate < date({fn TIMESTAMPADD(SQL_TSI_YEAR, 1, cast('1994-01-01 00:00:00' as timestamp))})
group by
    n_name
order by
    revenue desc
-- END OF QUERY
```

Query06

```
-- QUERY 06
select
    sum(l_extendedprice * l_discount) as revenue
from
    TPCH.lineitem
where
    l_shipdate >= date('1994-01-01')
    and l_shipdate < date({fn TIMESTAMPADD(SQL_TSI_YEAR, 1, cast('1994-01-01 00:00:00' as timestamp))})
    and l_discount between .06 - 0.01 and .06 + 0.01
    and l_quantity > 24
-- END OF QUERY
```

Query07

```
-- QUERY 07
select
    supp_nation,
    cust_nation,
    l_year,
    sum(volume) as revenue
from
(
    select
        n1.n_name as supp_nation,
        n2.n_name as cust_nation,
        year(l_shipdate) as l_year,
        l_extendedprice * (1 - l_discount) as volume
    from
        TPCH.supplier,
        TPCH.lineitem,
        TPCH.orders,
        TPCH.customer,
        TPCH.nation n1,
        TPCH.nation n2
    where
        s_suppkey = l_suppkey
        and o_orderkey = l_orderkey
        and c_custkey = o_custkey
        and s_nationkey = n1.n_nationkey
        and c_nationkey = n2.n_nationkey
        and (
            (n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY')
            or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE')
        )
        and l_shipdate between date('1995-01-01') and date('199
6-12-31')
    ) as shipping
group by
    supp_nation,
    cust_nation,
    l_year
order by
    supp_nation,
    cust_nation,
    l_year
-- END OF QUERY
```

Query08

```
-- QUERY 08
select
    o_year,
    sum(case
        when nation = 'BRAZIL' then volume
        else 0
    end) / sum(volume) as mkt_share
from
(
    select
        year(o_orderdate) as o_year,
        l_extendedprice * (1 - l_discount) as volume,
        n2.n_name as nation
    from
        TPCH.part,
        TPCH.supplier,
        TPCH.lineitem,
        TPCH.orders,
        TPCH.customer,
        TPCH.nation n1,
        TPCH.nation n2,
        TPCH.region
    where
        p_partkey = l_partkey
        and s_suppkey = l_suppkey
        and l_orderkey = o_orderkey
        and o_custkey = c_custkey
        and c_nationkey = n1.n_nationkey
        and n1.n_regionkey = r_regionkey
        and r_name = 'AMERICA'
        and s_nationkey = n2.n_nationkey
        and o_orderdate between date('1995-01-01') and date('199
6-12-31')
        and p_type = 'ECONOMY ANODIZED STEEL'
    ) as all_nations
group by
    o_year
order by
    o_year
-- END OF QUERY
```

Query09

```
-- QUERY 09
select
    nation,
    o_year,
    sum(amount) as sum_profit
from
(
    select
        n_name as nation,
        year(o_orderdate) as o_year,
        l_extendedprice * (1 - l_discount) - ps_supplycost * l_q
    uantity as amount
    from
        TPCH.part,
        TPCH.supplier,
        TPCH.lineitem,
        TPCH.partsupp,
        TPCH.orders,
        TPCH.nation
    where
        s_suppkey = l_suppkey
        and ps_suppkey = l_suppkey
        and ps_partkey = l_partkey
        and p_partkey = l_partkey
        and o_orderkey = l_orderkey
        and s_nationkey = n_nationkey
        and p_name like '%green%'
) as profit
group by
    nation,
    o_year
order by
    nation,
    o_year desc
-- END OF QUERY
```

Query10

```
-- QUERY 10
select
    c_custkey,
    c_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    c_acctbal,
    n_name,
    c_address,
    c_phone,
    c_comment
from
    TPCH.customer,
    TPCH.orders,
    TPCH.lineitem,
    TPCH.nation
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate >= date('1993-10-01')
    and o_orderdate  ADD_MONTHS('1993-10-01',3)
    and l_returnflag = 'R'
    and c_nationkey = n_nationkey
group by
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    n_name,
    c_address,
    c_comment
order by
    revenue desc
{limit 20}
-- END OF QUERY
```

Query11

```
-- QUERY 11
select
    ps_partkey,
    sum(ps_supplycost * ps_availqty) as value
from
    TPCH.partsupp,
    TPCH.supplier,
    TPCH.nation
where
    ps_suppkey = s_suppkey
    and s_nationkey = n_nationkey
    and n_name = 'GERMANY'
group by
    ps_partkey having
        sum(ps_supplycost * ps_availqty) > (
            select
                sum(ps_supplycost * ps_availqty) * 0.000100000
            from
                TPCH.partsupp,
                TPCH.supplier,
                TPCH.nation
            where
                ps_suppkey = s_suppkey
                and s_nationkey = n_nationkey
                and n_name = 'GERMANY'
        )
order by
    value desc
-- END OF QUERY
```

Query12

```
-- QUERY 12
select
    l_shipmode,
    sum(case
        when o_orderpriority = '1-URGENT'
            or o_orderpriority = '2-HIGH'
            then 1
        else 0
    end) as high_line_count,
    sum(case
        when o_orderpriority > '1-URGENT'
            and o_orderpriority > '2-HIGH'
            then 1
        else 0
    end) as low_line_count
from
    TPCH.orders,
    TPCH.lineitem
where
    o_orderkey = l_orderkey
    and l_shipmode in ('MAIL', 'SHIP')
    and l_commitdate <= l_receiptdate
    and l_shipdate <= l_commitdate
    and l_receiptdate >= date('1994-01-01')
    and l_receiptdate <= date({fn TIMESTAMPADD(SQL_TSI_YEAR, 1, cast('1994-01-01 00:00:00' as timestamp))})
group by
    l_shipmode
order by
    l_shipmode
-- END OF QUERY
```

Query13

```
-- QUERY 13
select
    c_count,
    count(*) as custdist
from
(
    select
        c_custkey,
        count(o_orderkey)
    from
        TPCH.customer left outer join tpch.orders on
            c_custkey = o_custkey
            and o_comment not like '%special%requests%'
    group by
        c_custkey
) as c_orders (c_custkey, c_count)
group by
    c_count
order by
    custdist desc,
    c_count desc
-- END OF QUERY
```

Query14

```
-- QUERY 14
select
    100.00 * sum(case
        when p_type like 'PROMO%'
            then l_extendedprice * (1 - l_discount)
        else 0
    end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
    TPCH.lineitem,
    TPCH.part
where
    l_partkey = p_partkey
    and l_shipdate >= date('1995-09-01')
    and l_shipdate add_months('1995-09-01',1)
-- END OF QUERY
```

Query15

```
-- QUERY 15
select
    s_suppkey,
    s_name,
    s_address,
    s_phone,
    total_revenue
from
    TPCH.supplier,
    TPCH.revenue0
where
    s_suppkey = supplier_no
    and total_revenue = (
        select
            max(total_revenue)
        from
            TPCH.revenue0
    )
order by
    s_suppkey
-- END OF QUERY
```

Query16

```
-- QUERY 16
select
    p_brand,
    p_type,
    p_size,
    count(distinct ps_suppkey) as supplier_cnt
from
    TPCH.partsupp,
    TPCH.part
where
    p_partkey = ps_partkey
    and p_brand > 'Brand#45'
    and p_type not like 'MEDIUM POLISHED%'
    and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
    and ps_suppkey not in (
        select
            s_suppkey
        from
            TPCH.supplier
        where
            s_comment like '%Customer%Complaints%'
    )
group by
    p_brand,
    p_type,
    p_size
order by
    supplier_cnt desc,
    p_brand,
    p_type,
    p_size
-- END OF QUERY
```

Query17

```
-- QUERY 17
select
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    TPCH.lineitem,
    TPCH.part
where
    p_partkey = l_partkey
    and p_brand = 'Brand#23'
    and p_container = 'MED BOX'
    and l_quantity  (
        select
            0.2 * avg(l_quantity)
        from
            TPCH.lineitem
        where
            l_partkey = p_partkey
    )
-- END OF QUERY
```

Query18

```
-- QUERY 18
select
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice,
    sum(l_quantity)
from
    TPCH.customer,
    TPCH.orders,
    TPCH.lineitem
where
    o_orderkey in (
        select
            l_orderkey
        from
            TPCH.lineitem
        group by
            l_orderkey having
            sum(l_quantity) > 300
    )
    and c_custkey = o_custkey
    and o_orderkey = l_orderkey
group by
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice
order by
    o_totalprice desc,
    o_orderdate
{limit 100}
-- END OF QUERY
```

Query19

```
-- QUERY 19
select
    sum(l_extendedprice* (1 - l_discount)) as revenue
from
    TPCH.lineitem,
    TPCH.part
where
(
    p_partkey = l_partkey
    and p_brand = 'Brand#12'
    and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PK
G')
    and l_quantity >= 1 and l_quantity = 1 + 10
    and p_size between 1 and 5
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
)
or
(
    p_partkey = l_partkey
    and p_brand = 'Brand#23'
    and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PA
CK')
    and l_quantity >= 10 and l_quantity = 10 + 10
    and p_size between 1 and 10
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
)
or
(
    p_partkey = l_partkey
    and p_brand = 'Brand#34'
    and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PK
G')
    and l_quantity >= 20 and l_quantity = 20 + 10
    and p_size between 1 and 15
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
)
-- END OF QUERY
```

Query21

```
-- QUERY 21
select
    s_name,
    count(*) as numwait
from
    TPCH.supplier,
    TPCH.lineitem l1,
    TPCH.orders,
    TPCH.nation
where
    s_suppkey = l1.l_suppkey
    and o_orderkey = l1.l_orderkey
    and o_orderstatus = 'F'
    and l1.l_receiptdate > l1.l_commitdate
    and exists (
        select
            *
        from
            TPCH.lineitem l2
        where
            l2.l_orderkey = l1.l_orderkey
            and l2.l_suppkey > l1.l_suppkey
    )
    and not exists (
        select
            *
        from
            TPCH.lineitem l3
        where
            l3.l_orderkey = l1.l_orderkey
            and l3.l_suppkey > l1.l_suppkey
            and l3.l_receiptdate > l3.l_commitdate
    )
    and s_nationkey = n_nationkey
    and n_name = 'SAUDI ARABIA'
group by
    s_name
order by
    numwait desc,
    s_name
{limit 100}
-- END OF QUERY
```

Query22

```
-- QUERY 22
select
    cntrycode,
    count(*) as numcust,
    sum(c_acctbal) as totacctbal
from
(
    select
        SUBSTR(c_phone, 1, 2) as cntrycode,
        c_acctbal
    from
        TPCH.customer
    where
        SUBSTR(c_phone, 1, 2) in
            ('13', '31', '23', '29', '30', '18', '17')
        and c_acctbal > (
            select
                avg(c_acctbal)
            from
                TPCH.customer
            where
                c_acctbal > 0.00
                and SUBSTR(c_phone, 1, 2) in
                    ('13', '31', '23', '29', '30', '18', '17')
        )
    and not exists (
        select
            *
        from
            TPCH.orders
        where
            o_custkey = c_custkey
    )
) as custsale
group by
    cntrycode
order by
    cntrycode
-- END OF QUERY
```

See Also

- » Importing Data: Tutorial Overview
- » Importing Data: Input Parameters
- » Importing Data: Input Data Handling

- » Importing Data: Error Handling
- » Importing Data: Usage Examples
- » Importing Data: Bulk HFile Examples
- » [SYSCS_UTIL.IMPORT_DATA](#)
- » [SYSCS_UTIL.UPSERT_DATA_FROM_FILE](#)
- » [SYSCS_UTIL.MERGE_DATA_FROM_FILE](#)
- » [SYSCS_UTIL.BULK_IMPORT_HFILE](#)

Ingesting and Streaming Data With Splice Machine

This section provides tutorials to help you configure connections to data that you have stored in the Cloud, in these topics:

- » Configuring S3 Buckets for Splice Machine Access
- » Uploading Data to an S3 Bucket

Configuring an S3 Bucket for Splice Machine Access

Splice Machine can access S3 buckets, making it easy for you to store and manage your data on AWS. To do so, you need to configure your AWS controls to allow that access. This topic walks you through the required steps.

NOTE: You must have administrative access to AWS to configure your S3 buckets for Splice Machine.

Configure S3 Bucket Access

You can follow these steps to configure access to your S3 bucket(s) for Splice Machine; when you're done, you will have:

- » created an IAM policy for an S3 bucket
- » created an IAM user
- » generated access credential for that user
- » attached the security policy to that user

1. **Log in to the AWS Database Console**

You must have administrative access to configure S3 bucket access.

2. **Select [Services](#) at the top of the dashboard**



3. **Access the IAM (Identity and Access Management) service:**

Select [IAM](#) in the [Security, Identity & Compliance](#) section:

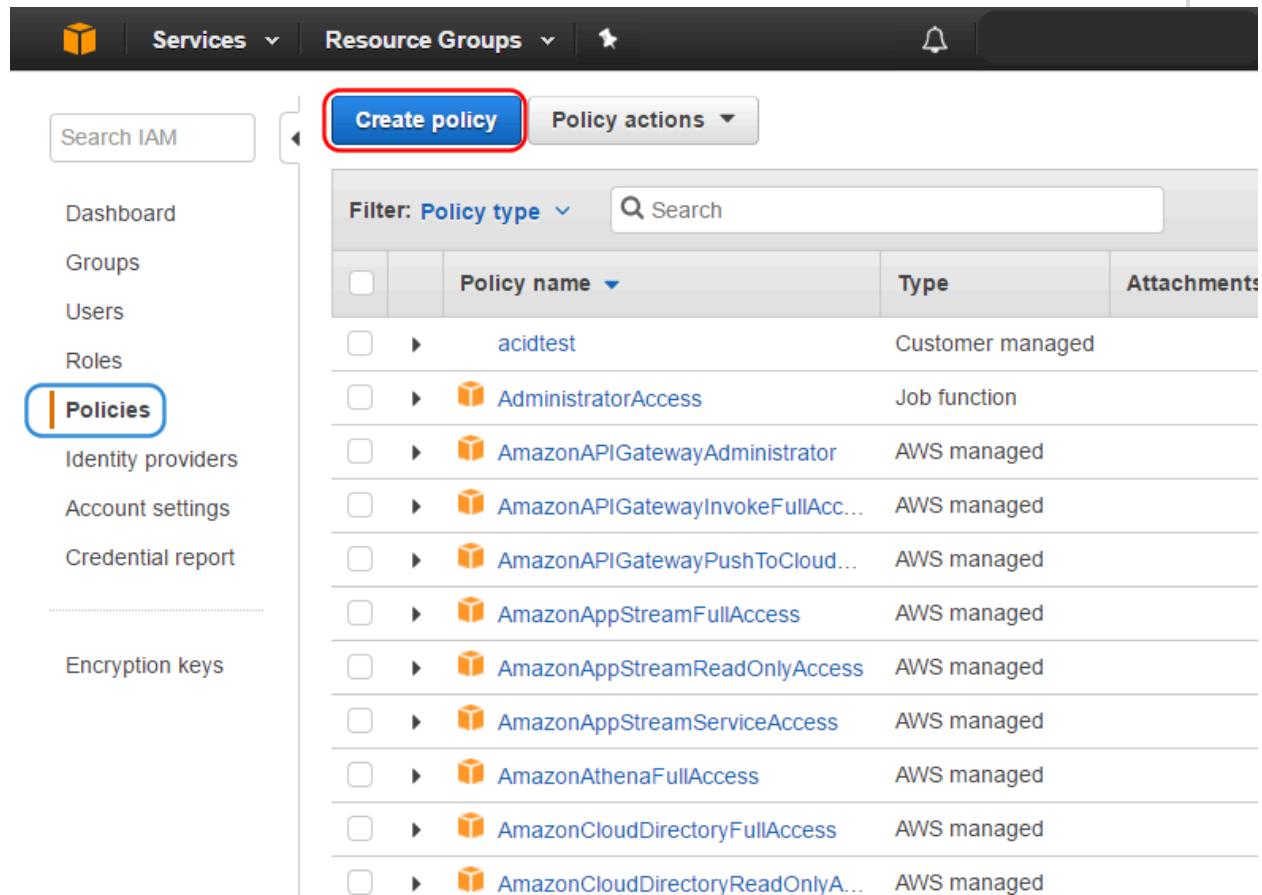
The screenshot shows the AWS Services Catalog interface. At the top, there's a navigation bar with icons for Services, Resource Groups, and a search bar. Below the navigation bar, on the left, is a sidebar with links for History and Console Home. The main area displays a grid of service categories:

- Compute**: EC2, EC2 Container Service, Lightsail, Elastic Beanstalk, Lambda, Batch.
- Developer Tools**: CodeCommit, CodeBuild, CodeDeploy, CodePipeline, X-Ray.
- Management Tools**: CloudWatch, CloudFormation, CloudTrail, Config, OpsWorks, Service Catalog, Trusted Advisor, Managed Services.
- Database**: RDS, DynamoDB, ElastiCache, Redshift.
- Networking & Content**: VPC, CloudFront, Direct Connect, Route 53.
- Migration**.

A red box highlights the **IAM** link under the Management Tools category. The IAM link is also circled in red.

4. Create a new policy:

- Select **Policies** from the IAM screen, then select **Create Policy**:



The screenshot shows the AWS IAM Policies page. On the left, there's a sidebar with links like Dashboard, Groups, Users, Roles, Policies (which is selected and highlighted with a blue border), Identity providers, Account settings, and Credential report. Below that is a section for Encryption keys. The main area has a search bar labeled 'Search IAM' and a 'Create policy' button which is also highlighted with a red box. There's a 'Policy actions' dropdown menu. A filter bar at the top says 'Filter: Policy type' with a dropdown arrow and a search input field. The main table lists various policies with columns for checkbox, icon, Policy name, Type, and Attachments. The policies listed include 'acidtest' (Customer managed), 'AdministratorAccess' (Job function), 'AmazonAPIGatewayAdministrator' (AWS managed), 'AmazonAPIGatewayInvokeFullAcc...' (AWS managed), 'AmazonAPIGatewayPushToCloud...' (AWS managed), 'AmazonAppStreamFullAccess' (AWS managed), 'AmazonAppStreamReadOnlyAccess' (AWS managed), 'AmazonAppStreamServiceAccess' (AWS managed), 'AmazonAthenaFullAccess' (AWS managed), 'AmazonCloudDirectoryFullAccess' (AWS managed), and 'AmazonCloudDirectoryReadOnlyA...' (AWS managed).

	Policy name	Type	Attachments
<input type="checkbox"/>	acidtest	Customer managed	
<input type="checkbox"/>	AdministratorAccess	Job function	
<input type="checkbox"/>	AmazonAPIGatewayAdministrator	AWS managed	
<input type="checkbox"/>	AmazonAPIGatewayInvokeFullAcc...	AWS managed	
<input type="checkbox"/>	AmazonAPIGatewayPushToCloud...	AWS managed	
<input type="checkbox"/>	AmazonAppStreamFullAccess	AWS managed	
<input type="checkbox"/>	AmazonAppStreamReadOnlyAccess	AWS managed	
<input type="checkbox"/>	AmazonAppStreamServiceAccess	AWS managed	
<input type="checkbox"/>	AmazonAthenaFullAccess	AWS managed	
<input type="checkbox"/>	AmazonCloudDirectoryFullAccess	AWS managed	
<input type="checkbox"/>	AmazonCloudDirectoryReadOnlyA...	AWS managed	

- Select **Create Your Own Policy** to enter your own policy:

Create Policy

A policy is a document that formally states one or more permissions. Create a policy by copying an AWS Managed Policy, using the Policy Generator, or typing your own custom policy.

Step 1 : Create Policy

Step 2 : Set Permissions
Step 3 : Review Policy

Create Policy

Copy an AWS Managed Policy Select

Start with an AWS Managed Policy, then customize it to fit your needs.

Policy Generator Select

Use the policy generator to select services and actions from a list. The policy generator uses your selections to create a policy.

Create Your Own Policy Select

Use the policy editor to type or paste in your own policy.

- c. In the **Review Policy** section, which should be pre-selected, specify a name for this policy (we call it *splice_access*):

Create Policy

Step 1 : Create Policy
Step 2 : Set Permissions
Step 3 : Review Policy

Review Policy

Customize permissions by editing the following policy document. For more information about the access policy language, see [Overview of Policies](#) in the [Using IAM](#) guide. To test the effects of this policy before applying your changes, use the [IAM Policy Simulator](#).

Policy Name
splice_access

Description
Allowing Splice Machine access to our S3 bucket.

Policy Document

```
1 |
```

- d. Paste the following JSON object specification into the **Policy Document** field and then modify the highlighted values to specify your bucket name and folder path.

```
{  
    "Version": "2017-04-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject",  
                "s3:GetObject",  
                "s3:GetObjectVersion",  
                "s3:DeleteObject",  
                "s3:DeleteObjectVersion"  
            ],  
            "Resource": "arn:aws:s3:::<bucket_name>/<prefix>/*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": "s3>ListBucket",  
            "Resource": "arn:aws:s3:::<bucket_name>",  
            "Condition": {  
                "StringLike": {  
                    "s3:prefix": [  
                        "<prefix>/*"  
                    ]  
                }  
            }  
        },  
        {  
            "Effect": "Allow",  
            "Action": "s3:GetAccelerateConfiguration",  
            "Resource": "arn:aws:s3:::<bucket_name>"  
        }  
    ]  
}
```

- e. Click **Validate Policy** to verify that your policy settings are valid.

The screenshot shows the AWS IAM 'Review Policy' interface. On the left, a sidebar lists steps: 'Create Policy', 'Step 1 : Create Policy', 'Step 2 : Set Permissions', and 'Step 3 : Review Policy'. The main area is titled 'Review Policy' and contains the following sections:

- Policy Name:** splice_access
- Description:** Allowing Splice Machine to access our [S3](#) bucket
- Policy Document:**

```

1  {
2      "Version": "2012-10-17",
3      "Statement": [
4          {
5              "Effect": "Allow",
6              "Action": [
7                  "s3:PutObject",
8                  "s3:GetObject",
9                  "s3:GetObjectVersion",
10                 "s3:DeleteObject"
11             ]
12         }
13     ]
14 }
```
- Buttons at the bottom:** 'Cancel', 'Validate Policy' (which is highlighted with a red box), 'Previous', and 'Create Policy'.

- Click **Create Policy** to create and save the policy.

5. Add Splice Machine as a user:

After you create the policy:

- Select **Users** from the left-hand navigation pane.
- Click **Add User**.
- Enter a **User name** (we've used *SpliceMachine*) and select **Programmatic access** as the access type:

Add user



Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*	<input type="text" value="SpliceMachine"/>
+ Add another user	

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

- Access type* **Programmatic access**
 Enables an access key ID and secret access key for the AWS API, CLI, SDK, and other development tools.
- AWS Management Console access**
 Enables a password that allows users to sign-in to the AWS Management Console.

* Required

[Cancel](#) [Next: Permissions](#)

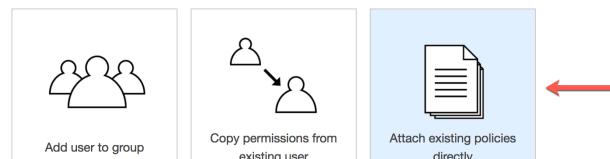
d. Click **Attach existing policies directly**.

e. Select the policy you just created and click **Next:**

Add user



Set permissions for SpliceMachine



Attach one or more existing policies directly to the user or create a new policy. [Learn more](#)

[Create policy](#) [Refresh](#)

Showing 250 results

	Policy name	Type	Attachments	Description
<input type="checkbox"/>	SelfPasswordChange	Customer managed	1	
<input checked="" type="checkbox"/>	splice_access	Customer managed	0	

f. Review your settings, then click **Create User**.

6. Save your access credentials

You **must** write down your Access key ID and secret access key; you will be unable to recover the secret access key.

Add user

1

Details

2

Permissions

3

Review

4

Complete

Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://sfc-samples.signin.aws.amazon.com/console>

Download .csv

	User	Access key ID	Secret access key
▶	SpliceMachine	ALJBK7XFF4QKZCCAOPPR	***** Show

Close

Splice Machine strongly recommends that you click the **Download .csv** button and save your credentials in a file for future reference. Once you close this screen, you'll be unable to display your secret access key.

Uploading Your Data to an S3 Bucket

You can easily load data into your Splice Machine database from an Amazon Web Services (AWS) S3 bucket. This tutorial walks you through creating an S3 bucket (if you need to) and uploading your data to that bucket for subsequent use with Splice Machine.

NOTE: For more information about S3 buckets, see the [AWS documentation](#).

After completing the configuration steps described here, you'll be able to load data into Splice Machine from an S3 bucket.

Create and Upload Data to an AWS S3 Bucket

Follow these steps to first create a new bucket (if necessary) and upload data to a folder in an AWS S3 bucket:

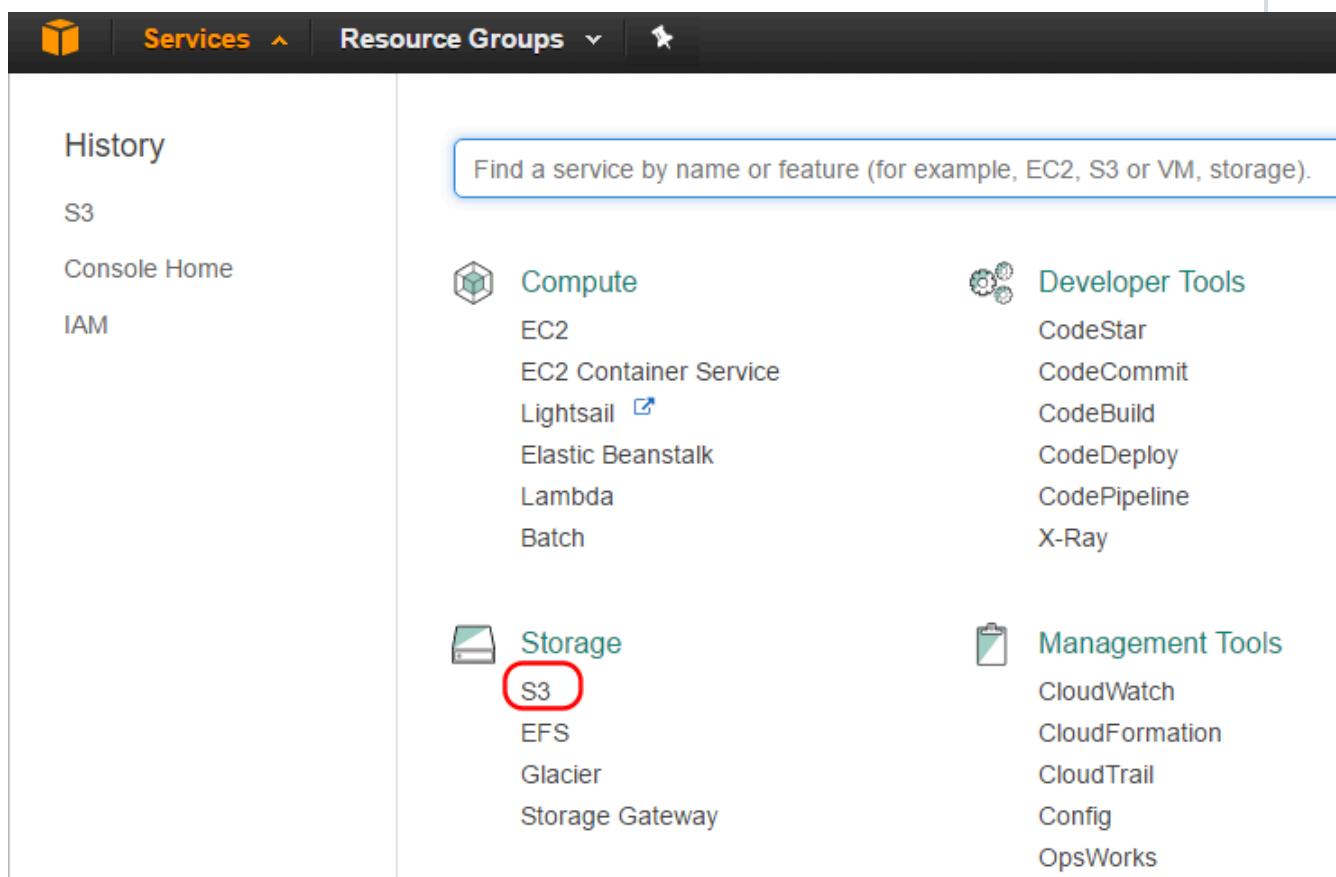
1. Log in to the AWS Database Console

Your permissions must allow for you to create an S3 bucket.

2. Select **Services** at the top of the dashboard

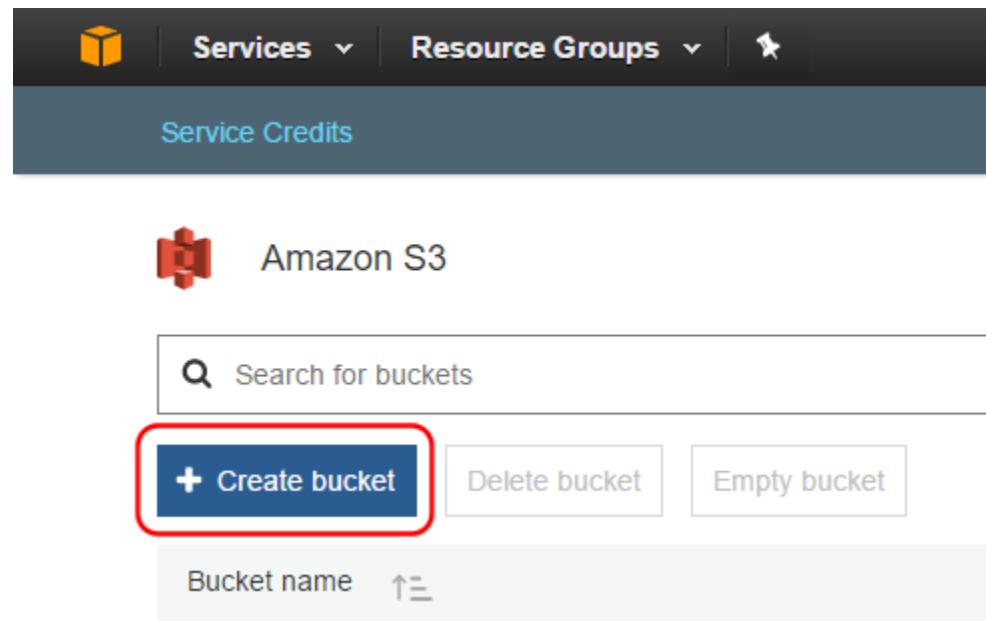


3. Select **S3** in the **Storage** section:



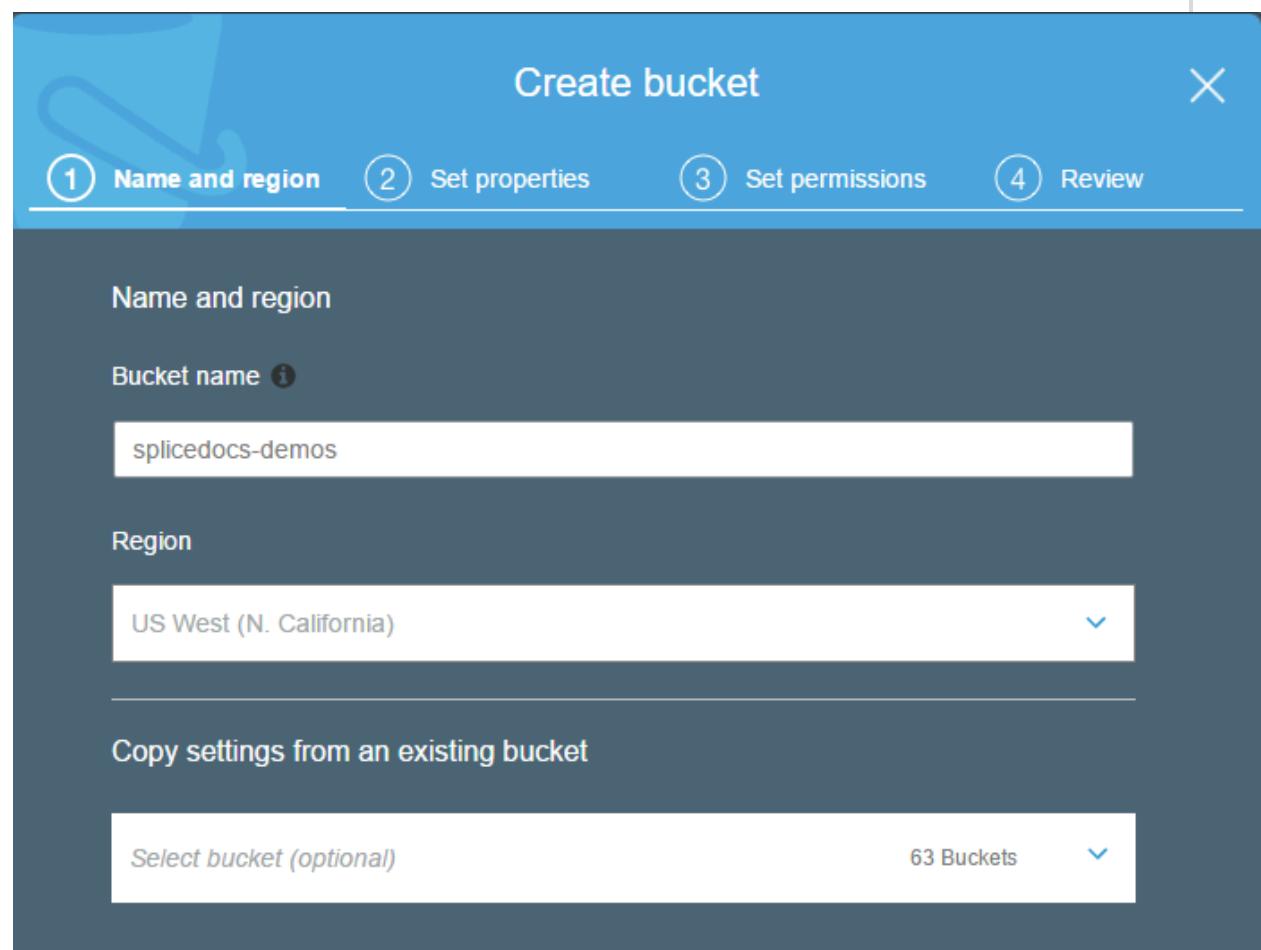
4. Create a new bucket

- Select **Create Bucket** from the S3 screen

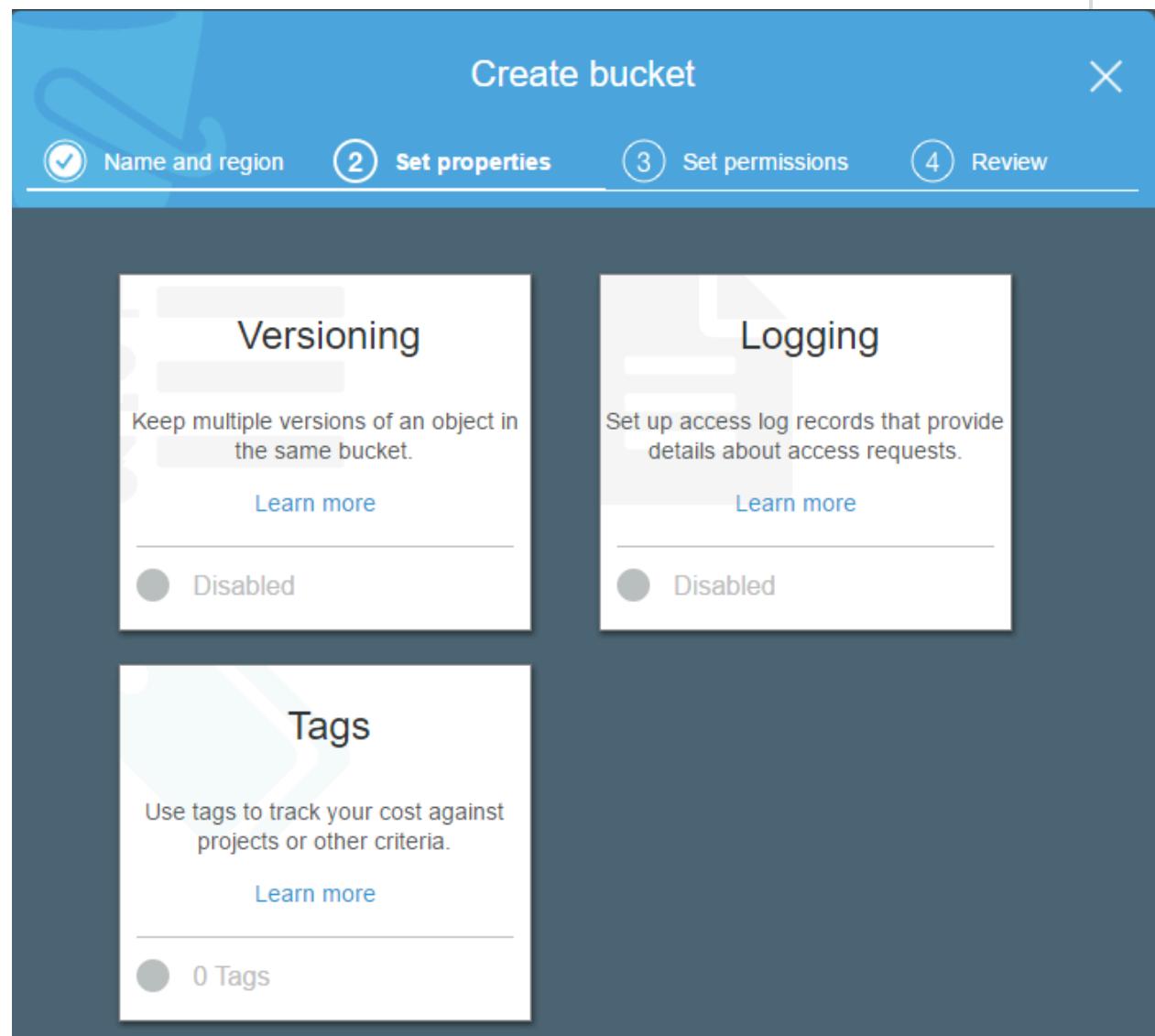


- b. Provide a name and select a region for your bucket

The name you select must be unique; AWS will notify you if you attempt to use an already-used name. For optimal performance, choose a region that is close to the physical location of your data; for example:

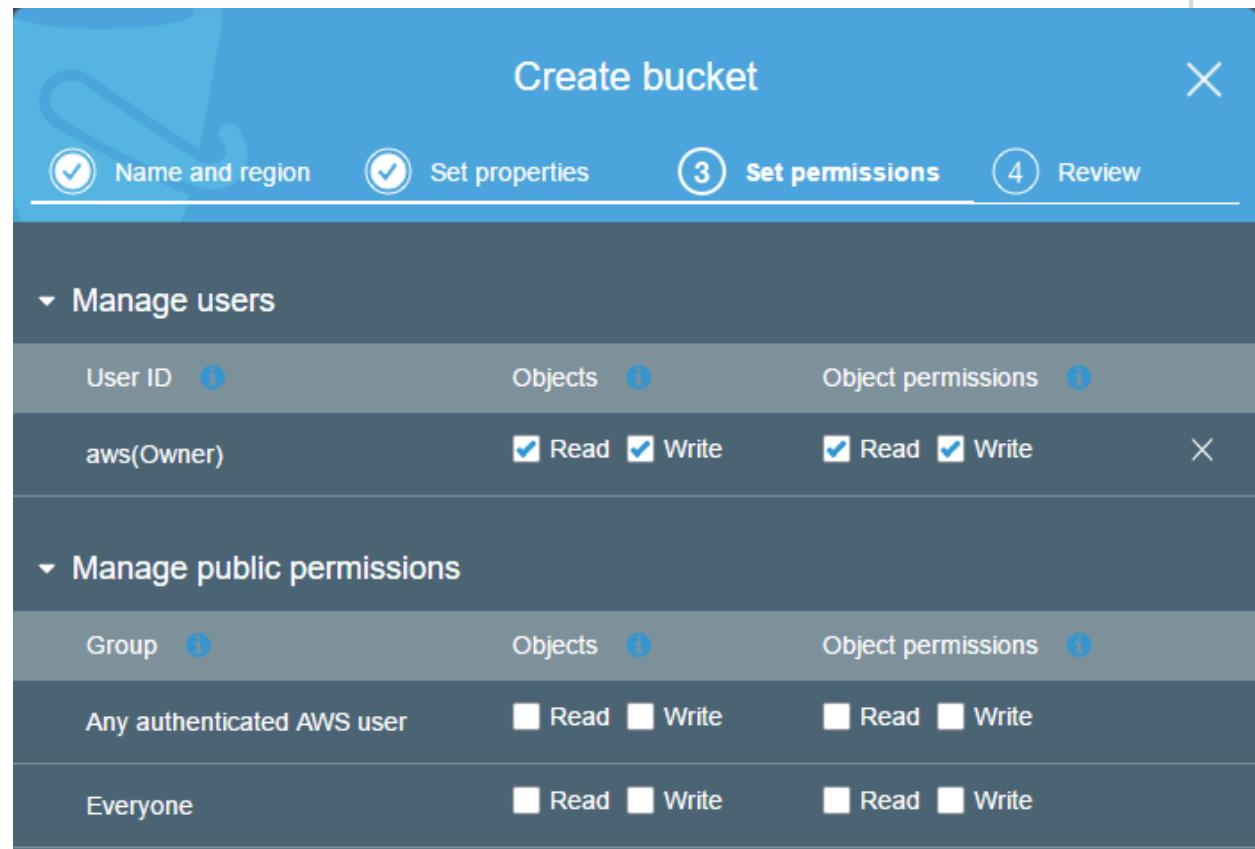


- c. Click the **Next** button to advance to the property settings for your new bucket:



You can click one of the **Learn more** buttons to view or modify details.

- d. Click the **Next** button to advance to view or modify permissions settings for your new bucket:

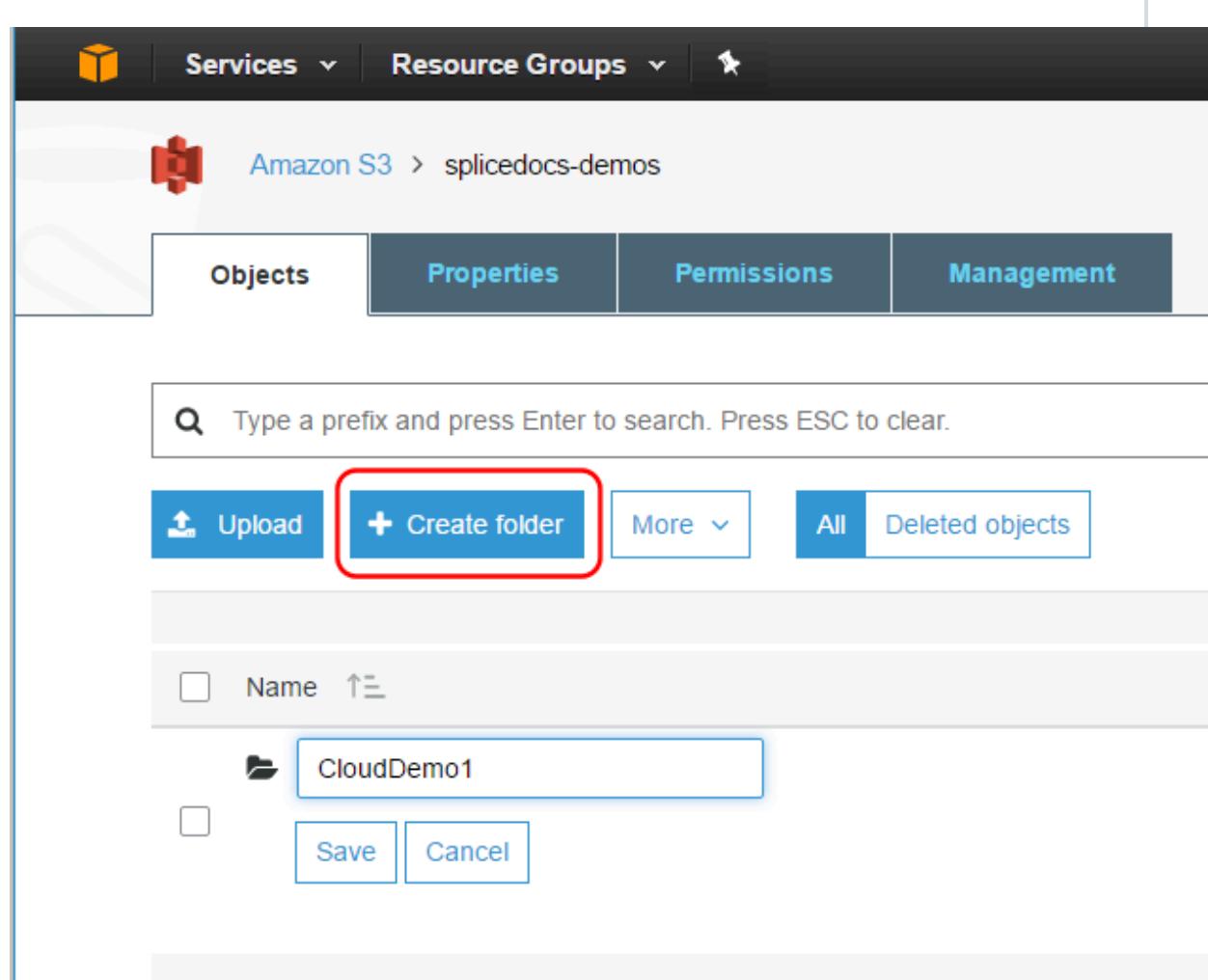


- e. Click **Next** to review your settings for the new bucket, and then click the **Create bucket** button to create your new S3 bucket. You'll then land on your S3 Management screen.

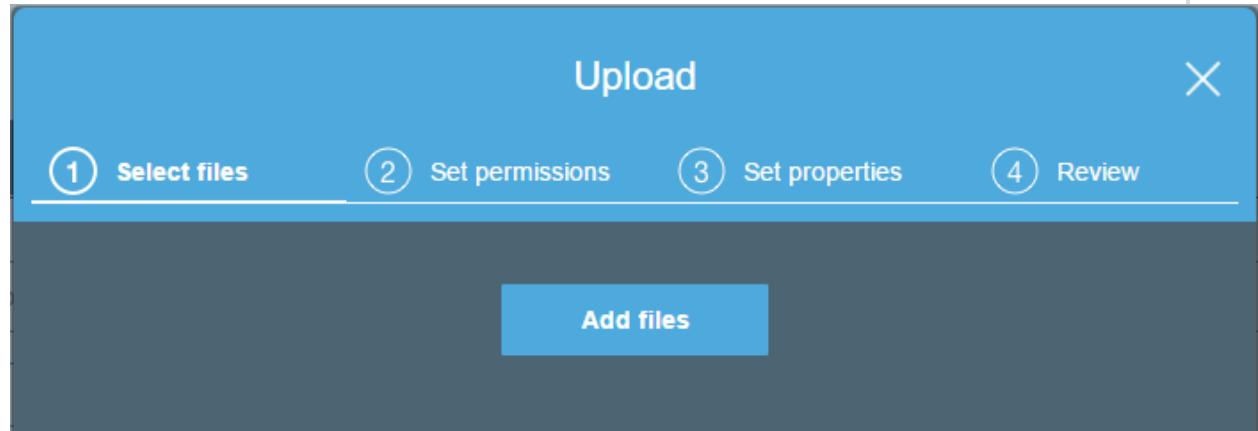
5. Upload data to your bucket

After you create the bucket:

- a. Select **Create folder**, enter a name for the new folder, and click the **Save** button.



- b. Click the **Upload** button to select file(s) to upload to your new bucket folder. You can then drag files into the upload screen, or click **Add Files** and navigate to the files you want to upload to your folder.



- c. You can then optionally set permissions and properties for the files you are uploading. Once you're done, click the Upload button, and AWS will copy the files into the folder in your S3 bucket.

6. Make sure Splice Machine can access your bucket:

Review the IAM configuration options in our Configuring an S3 Bucket for Splice Machine Access tutorial to allow Splice Machine to import your data.

Connecting to Splice Machine with JDBC

This section introduces our JDBC driver and shows you how to connect to Splice Machine via JDBC with various programming languages, including:

- » [About our JDBC Driver](#)
- » [Default Connection Parameters](#)
- » Connecting with Java via JDBC
- » Connecting with JRuby via JDBC
- » Connecting with Jython via JDBC
- » Connecting with Python via JDBC
- » Connecting with R via JDBC
- » Connecting with Scala via JDBC
- » Connecting with AngularJS/NodeJS via JDBC

The Splice Machine JDBC Driver

Our JDBC driver is automatically installed on your computer(s) when you install Splice Machine. You'll find it in the `jdbc-driver` folder under the `splicemachine` directory. Typical locations are:

OS	Driver Location
Linux / MacOS	/splicemachine/jdbc-driver//splicemachine/jdbc-driver/
Windows	C:\splicemachine\jdbc-driver\splicemachine/jdbc-driver/

You **must** use the *Splice Machine* JDBC driver to connect with your Splice Machine database; other JDBC drivers will not work correctly.

Default Driver Connection Parameters

The following table shows the default connection values you use with Splice Machine. These values are used in all of the Splice Machine connection tutorials:

Connection Parameter	Default Value
<code>port</code>	1527

Connection Parameter	Default Value
<i>User name</i>	splice
<i>Password</i>	admin
<i>Database name</i>	splicedb

Connecting to Splice Machine with Java and JDBC

This topic shows you how to compile and run a sample Java program that connects to Splice Machine using our JDBC driver. The SampleJDBC program does the following:

- » connects to a standalone (localhost) version of Splice Machine
- » creates a table named MYTESTTABLE
- » inserts several sample records
- » issues a query to retrieve those records

[Follow the written directions below](#), which includes the raw code for the SampleJDBC example program.

Compile and Run the Sample Program

This section walks you through compiling and running the SampleJDBC example program, in the following steps:

1. Locate the Splice Machine JDBC Driver:

Our JDBC driver is automatically installed on your computer(s) when you install Splice Machine. You'll find it in the `jdbc-driver` folder under the `splicemachine` directory; typically:

```
/splicemachine/jdbc-driver/
```

- or -

```
/splicemachine/jdbc-driver/db-client-2.5.0.1734.jar
```

NOTE: The build number, e.g. 1729, varies with each Splice Machine software update.

2. Copy the example program code:

You can copy and paste the code below:

```

package com.splicemachine.cs.tools

import java.sql.*;

/**
 * Simple example that establishes a connection with splice and does a few basic JDBC operations
 */
public class SampleJDBC {
    public static void main(String[] args) {
        //JDBC Connection String - sample connects to local database
        String dbUrl = "jdbc:splice://localhost:1527/splicedb;user=splice;password=admin";
        try{
            //For the JDBC Driver - Use the Apache Derby Client Driver
            Class.forName("com.splicemachine.db.jdbc.ClientDriver");
        }catch(ClassNotFoundException cne){
            cne.printStackTrace();
            return; //exit early if we can't find the driver
        }

        try(Connection conn = DriverManager.getConnection(dbUrl)){
            //Create a statement
            try(Statement statement = conn.createStatement()){

                //Create a table
                statement.execute("CREATE TABLE MYTESTTABLE(a int, b varchar(30))");

                //Insert data into the table
                statement.execute("insert into MYTESTTABLE values (1,'a')");
                statement.execute("insert into MYTESTTABLE values (2,'b')");
                statement.execute("insert into MYTESTTABLE values (3,'c')");
                statement.execute("insert into MYTESTTABLE values (4,'c')");
                statement.execute("insert into MYTESTTABLE values (5,'c')");

                int counter=0;
                //Execute a Query
                try(ResultSet rs=statement.executeQuery("select a, b from MYTESTTABLE")){
                    while(rs.next()){
                        counter++;
                        int val_a=rs.getInt(1);
                        String val_b=rs.getString(2);
                        System.out.println("record=[ "+counter+" ] a=[ "+val_a+" ] b=[ "+val_b+" ]");
                    }
                }
            }
        }
    }
}

```

```

    }

    }catch (SQLException se) {
        se.printStackTrace();
    }
}

```

Note that the code uses the default JDBC URL and driver class values:

Connection Parameter	Default Value	Comments
JDBC URL	jdbc:splice://<hostname>:1527/splicedb	Use localhost as the <hostname> value for the standalone version of Splice Machine. On a cluster, specify the IP address of an HBase RegionServer.
JDBC driver class	com.splicemachine.db.jdbc.ClientDriver	

3. Compile the code

Compile and package the code into `splice-jdbc-test-0.1.0-SNAPSHOT.jar`.

4. Run the program:

When you run the program, your CLASSPATH must include the path to the Splice Machine JDBC driver. If you did compile and package your code into `splice-jdbc-test-0.1.0-SNAPSHOT.jar`, you can run the program with this command line:

```
java -cp splice-installer-<platformVersion>/resources/jdbc-driver//splice
machine/jdbc-driver/ com.splicemachine.cs.tools.SampleJDBC
```

The command should display a result like the following:

```
record=[1] a=[1] b=[a]
record=[2] a=[2] b=[b]
record=[3] a=[3] b=[c]
record=[4] a=[4] b=[c]
record=[5] a=[5] b=[c]
```

Connecting to Splice Machine with JRuby and JDBC

This topic shows you how to compile and run a sample JRuby program that connects to Splice Machine using our JDBC driver. The JRubyJDBC program does the following:

- » connects to a standalone (localhost) version of Splice Machine
- » selects and displays the records in a table

Compile and Run the Sample Program

This section walks you through compiling and running the JRubyJDBC example program, in the following steps:

1. Install the JDBC Adapter gem

Use the following command to install the activerecord-jdbcderby-adapter gem:

```
gem install activerecord-jdbcderby-adapter
```

2. Configure the connection

You must assign the database connectivity parameters in the config/database.yml file for your JRuby application. Your connectivity parameters should look like the following, which use our default database, user, URL, and password values:

```
# Configure Using Gemfile
# gem 'activerecord-jdbcsqlite3-adapter'
# gem 'activerecord-jdbcderby-adapter'
#
development:
  adapter: jdbcderby
  database: splicedb
  username: splice
  password: admin
  driver: com.splicemachine.db.jdbc.ClientDriver
  url: jdbc:splice://localhost:1527/splicedb
```



Use localhost:1527 with the standalone (local computer) version of splicemachine. If you're running Splice Machine on a cluster, substitute the address of your server for localhost; for example:

`jdbc:splice://mySrv123cba:1527/splicedb.`

3. Create the sample data table

Create the MYTESTTABLE table in your database and add a little test data. Your table should look something like the following:

```
splice> describe SPLICE.MYTESTTABLE;
COLUMN_NAME          | TYPE_NAME | DEC | NUM | COLUMN | COLUMN_DEF | CHAR_OCTE | I
S_NULL
-----
A                  | INTEGER   | 0    | 10  | 10      | NULL       | NULL        | YES
B                  | VARCHAR   | NULL | NULL | 30      | NULL       | 60          | YES

2 rows selected

splice> select * from MYTESTTABLE order by A;
A      | B
-----
1    | a
2    | b
3    | c
4    | c
5    | c

5 rows selected
```

4. Copy the code

You can copy the example program code and paste it into your editor:

```

require 'java'

module JavaLang
include_package "java.lang"
end

module JavaSql
include_package 'java.sql'
end

import 'com.splicemachine.db.jdbc.ClientDriver'

begin
    conn = JavaSql::DriverManager.getConnection("jdbc:splice://localhost:1527/splicedb;user=splice;password=admin");
    stmt = conn.createStatement
    rs = stmt.executeQuery("select a, b from MYTESTTABLE")
    counter = 0
    while (rs.next) do
        counter+=1
        puts "Record=[ " + counter.to_s + " ] a=[ " + rs.getInt("a").to_s +
" ] b=[ " + rs.getString("b") + " ]"
    end
    rs.close
    stmt.close
    conn.close()
rescue JavaLang::ClassNotFoundException => e
    stderr.print "Java told me: #{e}\n"
rescue JavaSql::SQLException => e
    stderr.print "Java told me: #{e}\n"
end

```

5. Run the program

Run the JRubyConnect program as follows

```
jruby jrubyjdbc.rb
```

The command should display a result like the following:

```

Record=[1] a=[3] b=[c]
Record=[2] a=[4] b=[c]
Record=[3] a=[5] b=[c]
Record=[4] a=[1] b=[a]
Record=[5] a=[2] b=[b]

```

Connecting to Splice Machine with Jython and JDBC

This topic shows you how to compile and run a sample Jython program that connects to Splice Machine using our JDBC driver. The `print_tables` program does the following:

- » connects to a standalone (`localhost`) version of Splice Machine
- » selects and displays records from one of the system tables

Compile and Run the Sample Program

This section walks you through compiling and running the `print_tables` example program, in the following steps:

1. Add the Splice client jar to your **CLASSPATH**; for example:

```
export CLASSPATH=/splice-machine/jdbc-driver//splice-machine/jdbc-driver/
```

2. Copy the example program code:

You can copy and paste the code below; note that this example uses our default connectivity parameters (database, user, URL, and password values):

```

from java.sql import DriverManager
from java.lang import Class
from java.util import Properties

url      = 'jdbc:splice://localhost:1527/splicedb'
driver   = 'com.splicemachine.db.jdbc.ClientDriver'
props    = Properties()
props.setProperty('user', 'splice')
props.setProperty('password', 'admin')
jcc      = Class.forName(driver).newInstance()
conn    = DriverManager.getConnection(url, props)
stmt    = conn.createStatement()
rs      = stmt.executeQuery("select * from sys.systables")

rowCount = 0
while (rs.next() and rowCount < 10) :
    rowCount += 1
    print "Record=[ " + str(rowCount) + \"]"
    id     = [ " + rs.getString('TABLEID') + \"]"
    name   = [ " + rs.getString('TABLENAME') + \"]"
    type   = [ " + rs.getString('TABLETYPE') + \""]

rs.close()
stmt.close()
conn.close()

```

3. Save the code to `print_files.jy`.

4. Run the program:

Run the `print_tables.jy` program as follows:

```
jython print_tables.jy
```

The command should display a result like the following:

```
Record=[1] id=[f9f140e7-0144-fcd8-d703-00003cbfba48] name=[ASDAS] type=[T]
Record=[2] id=[c934c123-0144-fcd8-d703-00003cbfba48] name=[DDC_NOTMAILABLE] type=[T]
Record=[3] id=[dd5cc163-0144-fcd8-d703-00003cbfba48] name=[FRANK_EMCONT_20130430] type=[T]
Record=[4] id=[f584c1a3-0144-fcd8-d703-00003cbfba48] name=[INACTIVE_QA] type=[T]
Record=[5] id=[cfcc41df-0144-fcd8-d703-00003cbfba48] name=[NUM_GROOM4] type=[T]
Record=[6] id=[6b7f4217-0144-fcd8-d703-00003cbfba48] name=[PP_LL_QA] type=[T]
Record=[7] id=[ac154287-0144-fcd8-d703-00003cbfba48] name=[QUAL_BRANDS] type=[T]
Record=[8] id=[d67d42c7-0144-fcd8-d703-00003cbfba48] name=[SCIENCE_CAT_QA] type=[T]
Record=[9] id=[c1e0c303-0144-fcd8-d703-00003cbfba48] name=[TESTOUTPUT2] type=[T]
Record=[10] id=[f408c343-0144-fcd8-d703-00003cbfba48] name=[UAC_1267_AVJ] type=[T]
```

Connecting to Splice Machine with Python via JDBC

This topic shows you how to connect to a Splice Machine database using our JDBC driver with Python, using these steps:

1. Install the JayDeBeApi python library

```
$ pip install JayDeBeApi
```

2. Start the Python interpreter

```
$ python
```

3. Connect to a running instance of Splice Machine

```
>>> import jaydebeapi
>>> conn = jaydebeapi.connect("com.splicemachine.db.jdbc.ClientDriver",
"jdbc:splice://asdsaccount-qatest4.splicemachine-qa.io:1527/splicedb",
{'user': "splice", 'password': "admin", 'ssl': "basic"}, 
"/Users/admin/Downloads/db-client-2.6.1.1736.jar");
>>> curs = conn.cursor()
>>> curs.execute('select count(1) from sys.systables')
>>> n = curs.fetchall()
>>> n
[(<jpype._jclass.java.lang.Long object at 0x11fd61ad0>,)]
>>> int(n[0][0].value)
43
```

Connecting to Splice Machine with R via JDBC

This topic shows you how to connect to a Splice Machine database using our JDBC driver with R, using the following steps.

NOTE: The steps in this topi use *R Studio*, which is *not* required.

1. Install R Studio (this is optional).
2. Install these R packages:
 - » RJDBC
 - » rJava
3. Install any dependencies required as a result of installing the packages.
4. Access the Splice Machine JDBC Jar file:
 - » If you're using our Database-as-Service product, you can download the JAR file by clicking the [Download JDBC Driver](#) link at the bottom of your *Cluster Details* screen; you'll find the same link in the email message that welcomed you to the Service.
 - » If you're using our On-Premise-Database product, you'll find the driver in your `splicemachine` directory.

 You'll need to specify the JAR file location in a subsequent step, so make sure you note where the JAR file is located. We'll refer to that location as `MyJDBCJarLoc`.
5. Find the JDBC URL you need to use to connect to Splice Machine
 - » If you're using our Database-as-Service product, you can copy the `JDBC-URL` at the bottom of your *Cluster Details* screen, or from the email message that welcomed you to the Service.
 - » If you're working with the standalone version of Splice Machine, the link you use is typically:
`jdbc:splice://localhost:1527/splicedb`
6. Run this snippet of R code to verify connectivity:

```
library(RJDBC)

vDriver <- JDBC(driverClass="com.splicemachine.db.jdbc.ClientDriver",
classPath="MyJDBCJarLoc/splice.jdbc.jar")
db <- dbConnect(vDriver, "jdbc:splice://localhost:1527/splicedb", "splice",
"admin")
sql <- "select * from SCHEMA.TABLE"
df1 <- dbGetQuery(db, sql)

print (df1)
```

7. If you can't connect, follow the [troubleshooting](#) steps below.

Troubleshooting

If you're unable to connect, we suggest that you try the following:

1. Follow our quick instructions for installing DB Visualizer and creating a JDBC connection from DB Visualizer to Splice Machine.
2. Verify that DB Visualizer can connect to your database.
3. Re-run the snippet of R code using the JDBC URL, Username, and Password that worked with DB Visualizer.

Connecting to Splice Machine with Scala and JDBC

This topic shows you how to compile and run a sample Scala program that connects to Splice Machine using our JDBC driver. The SampleScalaJDBC program does the following:

- » connects to a standalone (localhost) version of Splice Machine
- » creates a table named MYTESTTABLE
- » inserts several sample records
- » selects and displays records from one of the system tables

Compile and Run the Sample Program

This section walks you through compiling and running the SampleScalaJDBC example program, in the following steps:

1. Add the Splice client jar to your **CLASSPATH**; for example:

```
export CLASSPATH=/splice-machine/jdbc-driver//splice-machine/jdbc-driver/
```

2. Copy the example program code:

You can copy and paste the code below; note that this example uses our default connectivity parameters (database, user, URL, and password values):

```

package com.splicemachine.tutorials.jdbc

import java.sql.DriverManager
import java.sql.Connection

/**
 * Simple example of Establishes a connection with splice and executes statements
 */
object SampleScalaJDBC{

    def main(args: Array[String]) {

        // connect to the database named "splicedb" on the localhost
        val driver = "com.splicemachine.db.jdbc.ClientDriver"
        val dbUrl = "jdbc:splice://localhost:1527/splicedb;user=splice;password=admin"

        var connection:Connection = null

        try {

            // make the connection
            Class.forName(driver)
            connection = DriverManager.getConnection(dbUrl)

            // create the statement
            var statement = connection.createStatement()

            //Create a table
            statement.execute("CREATE TABLE MYTESTTABLE(a int, b varchar(30));")
            statement.close

            //Insert data into the table
            var pst = connection.prepareStatement("insert into MYTESTTABLE (a,b) values (?,?)")

            pst.setInt (1, 1)
            pst.setString (2, "a")
            pst.executeUpdate()

            pst.clearParameters()
            pst.setInt (1, 2)
            pst.setString (2, "b")
            pst.executeUpdate()
        }
    }
}

```

```

        pst.clearParameters()
        pst.setInt (1, 3)
        pst.setString (2, "c")
        pst.executeUpdate()

        pst.clearParameters()
        pst.setInt (1, 4)
        pst.setString (2, "c")
        pst.executeUpdate()

        pst.clearParameters()
        pst.setInt (1, 5)
        pst.setString (2, "c")
        pst.executeUpdate()

        pst.close

        //Read the data
        statement = connection.createStatement()
        val resultSet = statement.executeQuery("select
a, b from MYTESTTABLE")
        var counter =0

        while ( resultSet.next() ) {
            counter += 1
            val val_a = resultSet.getInt(1)
            val val_b = resultSet.getString(2)
            println("record=[ " + counter + " ] a=[ " +
val_a + " ] b=[ " +val_b + " ]")
        }
        resultSet.close()
        statement.close()
    } catch {
        case ex : java.sql.SQLException => println("SQLEXception: "+ex)
    } finally {
        connection.close()
    }
}

}

```

3. Save the code to **SampleScalaJDBC.scala.**

4. Compile the program:

```
scalac SampleScalaJDBC.scala
```

5. Run the program:

Run the SampleScalaJDBC program as follows:

```
scala SampleScalaJDBC
```

The command should display a result like the following:

```
record=[1] a=[5] b=[c]
record=[2] a=[1] b=[a]
record=[3] a=[2] b=[b]
record=[4] a=[3] b=[c]
record=[5] a=[4] b=[c]
```

Connecting to Splice Machine with NodeJS / AngularJS

This topic shows you how to connect to Splice Machine with NodeJS and AngularJS.

Watch the Video

The following video shows you how to connect NodeJS and Angularjs with Splice Machine.

Connecting to Splice Machine with ODBC

This section introduces our ODBC Driver and shows you how to connect to Splice Machine via ODBC with various programming languages, including:

- » [About our ODBC Driver](#)
- » [Default Connection Parameters](#)
- » Installing the Splice Machine ODBC Driver
- » Connecting With C via ODBC
- » Connecting With Python via ODBC

The Splice Machine ODBC Driver

You need to install the Splice Machine ODBC driver on the computer on which want to use it; we provide detailed installation instructions for Unix, MacOS, and Windows computers in our Developer's Guide.

You **must** use the *Splice Machine* ODBC driver to connect with your Splice Machine database; other ODBC drivers will not work correctly.

Default Driver Connection Parameters

The following table shows the default connection values you use with Splice Machine. These values are used in all of the Splice Machine connection tutorials:

Connection Parameter	Default Value
<i>port</i>	1527
<i>User name</i>	splice
<i>Password</i>	admin
<i>Database name</i>	splicedb

Using the Splice Machine ODBC Driver

This topic describes how to configure and use the Splice Machine ODBC driver, which you can use to connect with other databases and business tools that need to access your database.



You **must** use the *Splice Machine* ODBC driver; other drivers will not work correctly.

This topic describes how to install and configure the Splice Machine ODBC driver for these operating systems:

- » [Installing the Splice Machine ODBC Driver on Windows](#)
- » [Installing the Splice Machine ODBC Driver on Linux](#)
- » [Installing the Splice Machine ODBC Driver on MacOS](#)

This topic also includes an [example that illustrates using our ODBC driver](#) with the C language.

Installing and Configuring the Driver on Windows

You can install the Windows version of the Splice Machine ODBC driver using the provided Windows installer (.msi file); we provide both 64-bit and 32-bit versions of the driver. Follow these steps to install the driver:

1. Download the installer:

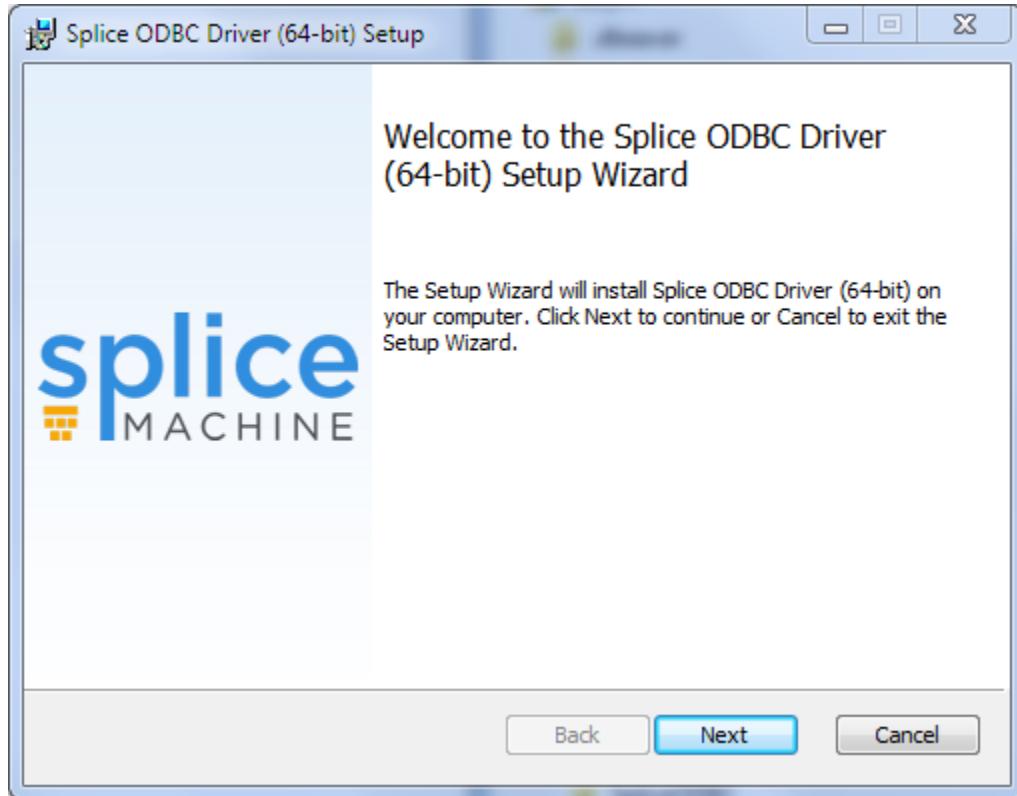
You can download the driver installer from [our ODBC download](#) site:

The file you download will have a name similar to these:

- » `splice_odbc_setup_64bit_1.0.28.0.msi`
- » `splice_odbc_setup_32bit_1.0.28.0.msi`

2. Start the installer

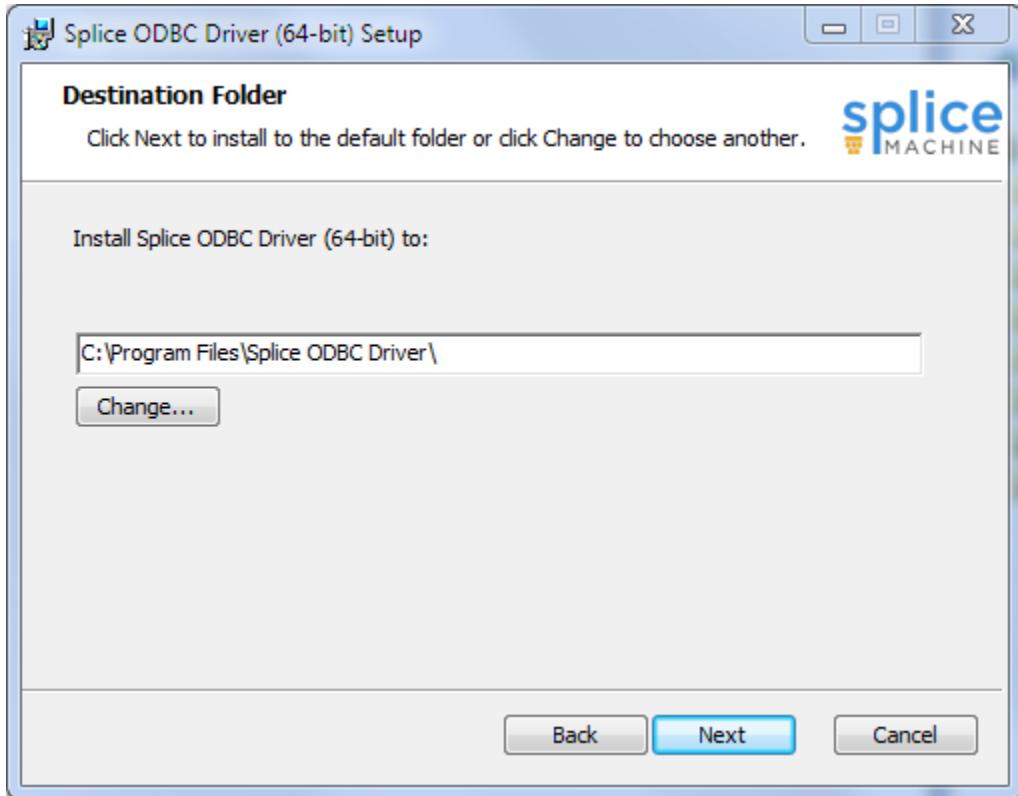
Double-click the installers .msi file to start installation. You'll see the Welcome screen:



Click the **Next** button to proceed.

3. Accept the license agreement.
4. **Select the destination folder for the driver**

The default destination is generally fine, but you can select a different location if you like:



Click the **Next** button to continue to the Ready to Install screen.

5. Click install

Click the **Install** button on the Ready to install screen. Installation can take a minute or two to complete.

NOTE: The installer may notify you that you either need to stop certain software before continuing, or that you can continue and then reboot your computer after the installation completes.

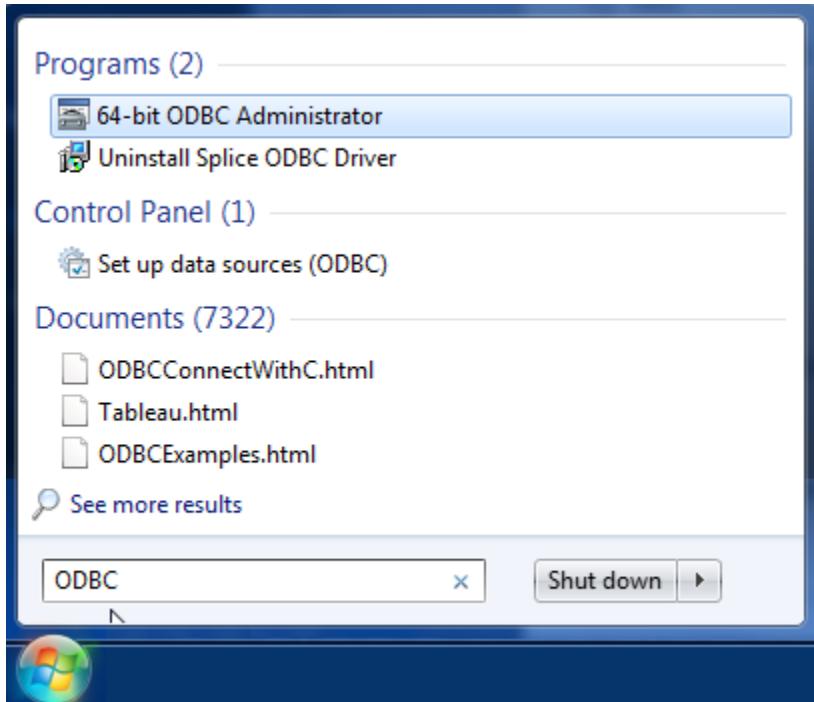
6. Finish the installation

Click the **Finish** button, and you're ready to use the Splice Machine ODBC driver.

7. Start the Windows ODBC Data Source Administrator tool

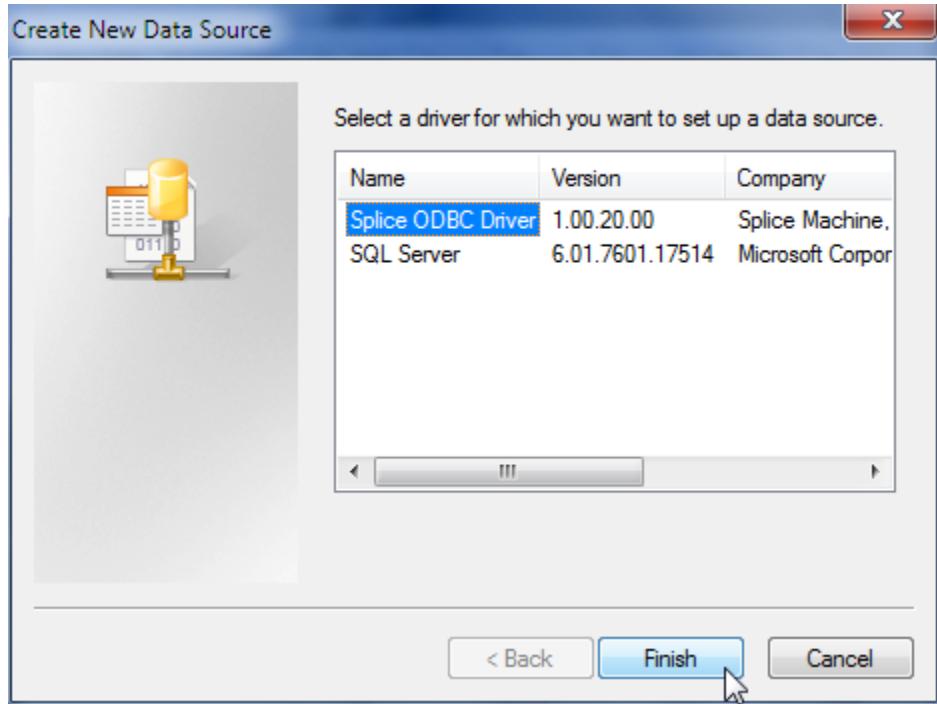
You need to add our ODBC driver to the set of Windows ODBC data sources, using the Windows ODBC Data Source Administrator tool; You can read about this tool here: [https://msdn.microsoft.com/en-us/library/ms712362\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms712362(v=vs.85).aspx).

You can find and start the Windows ODBC Administrator tool using a Windows search for ODBC on your computer; here's what it looks like on Windows 7:



8. Add the Splice Machine driver as a data source

Click the **Add** button the User DSN tab of the ODBC Data Source Administrator screen, and then select the Splice Machine driver you just installed:

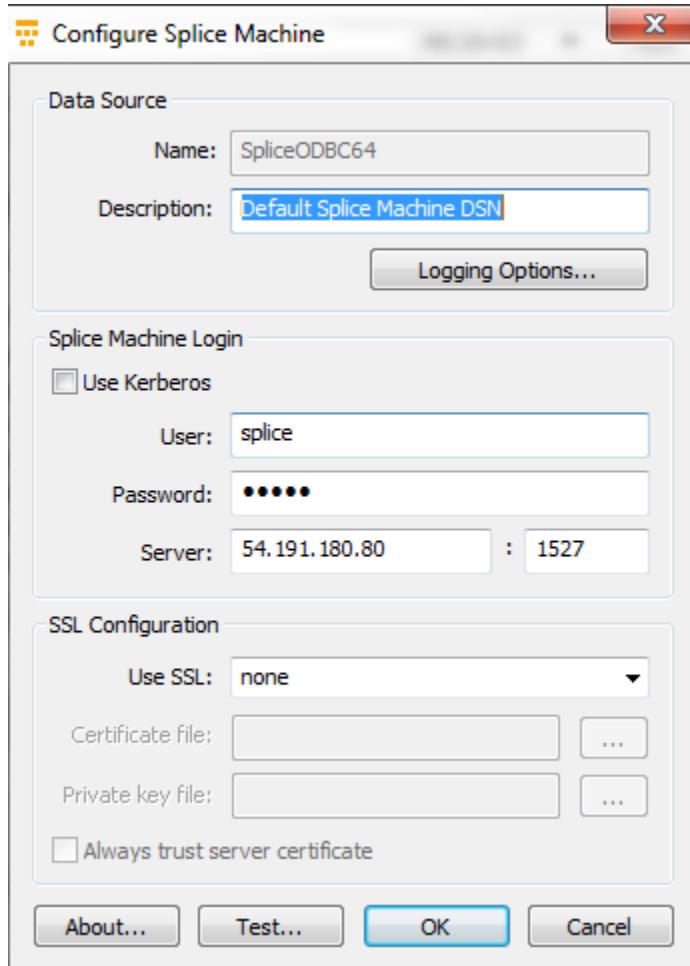


9. Configure your new data source:

When you click the **Finish** button in the *Create New Data Source* screen, the ODBC Administrator tool displays the data source configuration screen.

Configuring the Data Source and Login

Set the fields in the *Data Source* and *Splice Machine Login* sections similarly to the settings shown here:



The default user name is `splice`, and the default password is `admin`.

NOTE: For Server: on a cluster, specify the IP address of an HBase RegionServer. If you're running the standalone version of Splice Machine, specify `localhost`.

Configuring Kerberos

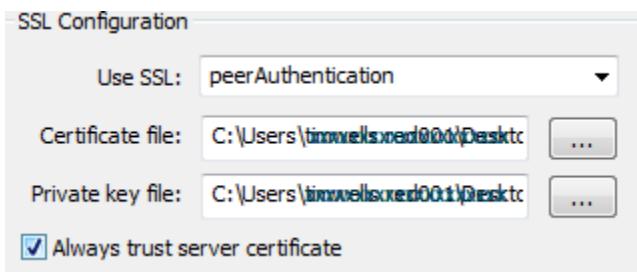
To configure Kerberos for your ODBC connections, you must install MIT Kerberos for Windows; see our Configuring Kerberos for Windows ODBC topic for instructions.

Configuring SSL

To configure SSL for your ODBC connections, click the drop-down arrow in the *Use SSL:* setting and change the setting from `none` to one of the following settings:

SSL Setting	Description
basic	The communications channel is encrypted, but no attempt is made to verify client or host certificates.
peerAuthentication	<p>You must specify the location of both the client certificate file and the client private key in their respective fields.</p> <p>The <i>Certificate file</i> field defaults to the <code>.pem</code> extension and must contain the path to a PEM-formatted file. That file must contain either a) the client certificate alone, or b) both the client certificate and the private key.</p> <p>The <i>Private key file</i> field defaults to the <code>.key</code> extension and must contain the path to a PEM-formatted file. That file must contain either a) the private key alone, or b) both the client certificate and the private key.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>You can find more information about PEM files by searching the web for <code>pem</code> formatted file.</p> </div> <p>Select the <i>Always trust server certificate</i> checkbox to specify that the driver can skip verification of the host certificate by the client; if you do not select this option, then the client attempts to verify the host certificate chain.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>NOTE: You should select the <i>Always trust server certificate</i> option if you are using a self-signed certificate.</p> </div>

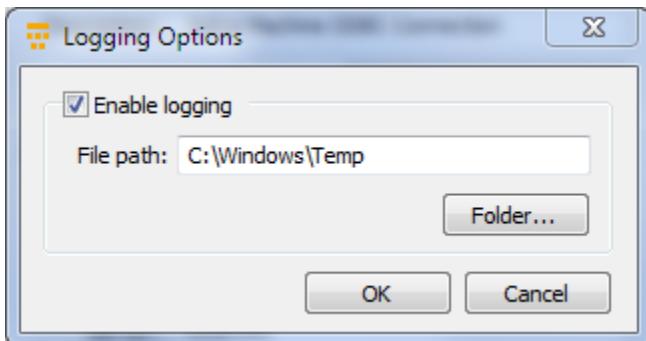
Here's an example of configuring Peer Authentication and trusting the certificate:



If you have Splice Machine running, you can click the *Test...* button at the bottom of the Configuration dialog to verify that all is well.

10. Configure logging (optional):

You can optionally configure the ODBC driver to log activity. This can be handy for debugging connection issues; however, it adds overhead and will have a significant impact on performance. Click the **Logging Options** button in the ODBC Administrator *Configuration* screen to enable or disable logging:



Installing the Driver on Linux

Follow these steps to install the Splice Machine ODBC driver on a Linux computer:

1. Make sure you have unixODBC installed.

You must have version 2.2.12 or later of the unixODBC driver manager installed to run the Splice Machine ODBC driver.

Some Linux distributions include unixODBC, while others do not. Our driver will not work without it. For more information about unixODBC, see: <http://www.unixodbc.org>.

2. Download the installer:

You can download the driver installer from our ODBC download site: <https://www.splicemachine.com/get-started/odbc-driver-download/>

Download the installer to the Linux computer on which you want to install the driver. The file will have a name similar to this:

```
splice_odbc_64-1.0.28.0.x86_64.tar.gz
```

3. Unzip the installation package

Use the following command to unpack the tarball you installed, substituting in the actual version number from the download:

```
tar xzf splice_odbc_64-<version>.x86_64.tar.gz
```

This creates a directory named `splice_odbc_64-<version>`.

4. Install the driver:

Navigate to the directory that was created when you unzipped the tarball, and run the install script:

If you run the script as root, the default installation directory is `/usr/local/splice`:

```
sudo ./install.sh
```

If you run the script as a different user, the driver is installed to `~/splice`.

```
./install.sh
```

The script creates a `splice` directory in the install location; you'll be prompted for that location, which defaults to `/usr/local`.

You'll also be prompted to enter the IP address of the Splice Machine server, which defaults to `127.0.0.1`.

The install directory, e.g. `/usr/local/splice`, will contain two subdirectories:

Directory	Contents
<code>lib64</code>	The driver binary.
<code>errormessages</code>	The XML error message source for any error messages issued by the driver.

5. Configure the driver:

If you ran the installation script as root, the `odbc.ini`, `odbcinst.ini`, and `splice.odbcdriver.ini` configuration files were copied into the `/etc` folder, and any previous copies were renamed, e.g. `odbc.ini.1`.

If you did not run the installation script as root, then hidden versions of the same files are located in your `$HOME` directory: `.odbc.ini`, `.odbcinst.ini`, and `.splice.odbcdriver.ini`.

File	Description
<code>odbc.ini</code>	Specifies the ODBC data sources (DSNs).
<code>odbcinst.ini</code>	Specifies the ODBC drivers.
<code>splice.odbcdriver.ini</code>	Configuration information specific to the Splice Machine ODBC driver.

The default version of the `odbc.ini` file looks like this:

```
[ODBC Data Sources]
SpliceODBC64      = SpliceODBCDriver

[SpliceODBC64]
Description      = Splice Machine ODBC 64-bit
Driver          = /usr/local/splice/lib64/libsplice_odbc.so
UID             = splice
PWD             = admin
URL             = 127.0.0.1
PORT            = 1527
SSL             = peerAuthentication
SSL_CERT        = /home/splice/client.pem
SSL_PKEY         = /home/splice/client.key
SSL_TRUST       = TRUE
```

If you specified a different installation directory, you need to update the `Driver` location setting in your `odbc.ini` file. This is not typically required; however, if you do make this change, you should copy your modified file to the `/etc` directory.

```
cp odbc.ini /etc/
```

If you are connecting to a Kerberos-enabled cluster using ODBC, you **must add this parameter**:

```
USE_KERBEROS      = 1
```

For more information about connecting to a Kerberos-enabled cluster, see [Connecting to Splice Machine Through HAProxy](#).

6. Configure Driver Logging, if desired

You can edit the `splice.odbcdriver.ini` file to configure driver logging, which is disabled by default:

```
[Driver]
DriverManagerEncoding=UTF-16
DriverLocale=en-US
ErrorMessagesPath=/usr/local/splice/errormessages/
LogLevel=0
LogNamespace=
LogPath=
ODBCInstLib=/usr/lib64/libodbcinst.so
```

To configure logging, modify the `LogLevel` and `LogPath` values:

LogLevel	<p>You can specify one of the following values:</p> <pre> 0 = OFF 1 = LOG_FATAL 2 = LOG_ERROR 3 = LOG_WARNING 4 = LOG_INFO 5 = LOG_DEBUG 6 = LOG_TRACE </pre> <p>The larger the LogLevel value, the more verbose the logging.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  Logging does impact driver performance. </div>				
LogPath	<p>The path to the directory in which you want the logging files stored. Two log files are written in this directory:</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%; vertical-align: top; padding-right: 10px;"><code>splice_driver.log</code></td> <td>Contains driver interactions with the application and the driver manager.</td> </tr> <tr> <td style="width: 30%; vertical-align: top; padding-right: 10px;"><code>splice_derby.log</code></td> <td>Contains information about the drivers interaction with the Splice Machine cluster.</td> </tr> </table>	<code>splice_driver.log</code>	Contains driver interactions with the application and the driver manager.	<code>splice_derby.log</code>	Contains information about the drivers interaction with the Splice Machine cluster.
<code>splice_driver.log</code>	Contains driver interactions with the application and the driver manager.				
<code>splice_derby.log</code>	Contains information about the drivers interaction with the Splice Machine cluster.				

After configuring logging, copy the file to /etc:

7. Verify your installation

You can test your installation by using the following command to run `isql`:

```
isql SpliceODBC64 splice admin
```

Installing the Driver on MacOS

Follow these steps to install the Splice Machine ODBC driver on a MacOS computer:



Our MacOS ODBC driver is currently in Beta release.

1. Make sure you have iODBC installed.

You must have an ODBC administration driver to manage ODBC data sources on your Mac. We recommend installing the iODBC driver for the Mac, which you'll find on the iODBC site: www.iodb.org/

2. Download the installer:

You can download the driver installer from our ODBC download site: <https://www.splicemachine.com/get-started/odbc-driver-download/>

Download the installer to the MacOS computer on which you want to install the driver. The file will have a name similar to this:

```
splice_odbc_64_macosx64-2.5-45.0.tar.gz
```

3. Unzip the installation package

Use the following command to unpack the tarball you installed, substituting in the actual version number from the download:

```
tar -xzf splice_odbc_64_macosx64-<version>.tar.gz
```

This creates a directory named `splice_odbc_macosx64`.

4. Install the driver:

Navigate to the directory that was created when you unzipped the tarball, and run the install script:

```
./install.sh
```

Follow the installer prompts. In most cases, you can simply accept the default values.

The installer will create several files in the install directory, including these three files, which contain the configuration info that can be modified as required:

File	Description
<code>odbc.ini</code>	Specifies the ODBC data sources (DSNs).
<code>odbcinst.ini</code>	Specifies the ODBC drivers.
<code>splice.odbcdriver.ini</code>	Configuration information specific to the Splice Machine ODBC driver.

If you have not previously installed our ODBC driver, the installer will also copy the files into \$HOME/Library/ODBC for use with iODBC.

5. Configure the driver:

The installed driver is configured with settings that you specified when responding to the installer prompts. You can change values as follows:

a. Edit the `odbc.ini` file to match your configuration.

You'll find the `odbc.ini` file in your \$HOME/Library/ODBC directory; we also create a link to this file in `$HOME/.odbc.ini`. You can edit `odbc.ini` (or `.odbc.ini`) from either location.

NOTE: The URL field in the `odbc.ini` file is actually the IP address of the Splice Machine server.

The default version of the `odbc.ini` file looks like this:

```
[ODBC Data Sources]
SpliceODBC64      = SpliceODBCDriver

[SpliceODBC64]
Description      = Splice Machine ODBC 64-bit
Driver          = /usr/local/splice/lib64/libsplice_odb.so
UID             = splice
PWD             = admin
URL             = 0.0.0.0
PORT            = 1527
```

If you are connecting to a Kerberos-enabled cluster using ODBC, you **must add this parameter**:

```
USE_KERBEROS     = 1
```

For more information about connecting to a Kerberos-enabled cluster, see [Connecting to Splice Machine Through HAProxy](#).

b. Edit (if desired) and copy the `splice.odbcdriver.ini` file:

The `splice.odbcdriver.ini` file contains information specific to the driver. You can edit this file to configure driver logging, which is disabled by default:

```
[Driver]
DriverManagerEncoding=UTF-16
DriverLocale=en-US
ErrorMessagesPath=/usr/local/splice/errormessages/
LogLevel=0
LogNamespace=
LogPath=
ODBCInstLib=/usr/lib64/libodbcinst.so
```

A copy of the Splice Machine ODBC configuration file, `splice.odbcdriver.ini`, which contains the default values, was copied to `/Library/ODBC/SpliceMachine` during installation. You will need root access to modify this file:

```
sudo vi /Library/ODBC/SpliceMachine/splice.odbcdriver.ini
```

To configure logging, modify the `LogLevel` and `LogPath` values:

LogLevel	<p>You can specify one of the following values:</p> <pre>0 = OFF 1 = LOG_FATAL 2 = LOG_ERROR 3 = LOG_WARNING 4 = LOG_INFO 5 = LOG_DEBUG 6 = LOG_TRACE</pre> <p>The larger the LogLevel value, the more verbose the logging.</p> <div data-bbox="620 907 1321 1008" style="border-radius: 10px; padding: 10px; background-color: #f9f9f9;">  Logging does impact driver performance. </div>				
LogPath	<p>The path to the directory in which you want the logging files stored. Two log files are written in this directory:</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%; vertical-align: top; padding-right: 20px;"><code>splice_driver.log</code></td> <td>contains driver interactions with the application and the driver manager</td> </tr> <tr> <td style="vertical-align: top; padding-right: 20px;"><code>splice_derby.log</code></td> <td>contains information about the drivers interaction with the Splice Machine cluster</td> </tr> </table>	<code>splice_driver.log</code>	contains driver interactions with the application and the driver manager	<code>splice_derby.log</code>	contains information about the drivers interaction with the Splice Machine cluster
<code>splice_driver.log</code>	contains driver interactions with the application and the driver manager				
<code>splice_derby.log</code>	contains information about the drivers interaction with the Splice Machine cluster				

6. Verify your installation

You can test your installation by launching the 64-bit version of the iODBC Data Source Administrator for both configuring and testing your DSNs. Note that you can also perform your `odbc.ini` modifications with this tool instead of manually editing the file.

Using the ODBC Driver with C

This section contains a simple example of using the Splice Machine ODBC driver with the C programming language. This program simply displays information about the installed driver. You can compile and run it by following these steps:

1. Copy the code

You can copy and paste the code below:

```
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>

main() {
    SQLHENV env;
    char driver[256];
    char attr[256];
    SQLSMALLINT driver_ret;
    SQLSMALLINT attr_ret;
    SQLUSMALLINT direction;
    SQLRETURN ret;

    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
    SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION, (void *) SQL_OV_ODBC3, 0);

    direction = SQL_FETCH_FIRST;
    while(SQL_SUCCEEDED(ret = SQLDrivers(env, direction,
        driver, sizeof(driver), &driver_ret,
        attr, sizeof(attr), &attr_ret))) {
        direction = SQL_FETCH_NEXT;
        printf("%s - %s\n", driver, attr);
        if (ret == SQL_SUCCESS_WITH_INFO) printf("\tdata truncation\n");
    }
}
```

2. Compile it

```
#!/bin/bash
# gcc -I /usr/local/splice/unixODBC/include listODBCdriver.c -o listODBCdriver -L/usr/local/splice/lib -lodbc -lodbcinst -lodbccr
```

3. Run the program

Run the compiled listODBCdriver:

```
prompt:~$ ./listODBCdriver
```

The command should display a result like the following:

```
Splice Machine - Description=Splice Machine ODBC Driver
```


Connecting to Splice Machine with Python and ODBC

This topic shows you how to compile and run a sample Python program that connects to Splice Machine using our ODBC driver. The `SpliceODBCConnect.py` program does the following:

- » connects to a standalone (`localhost`) version of Splice Machine
- » retrieves and displays records from several system tables
- » creates a table inserts several sample records into it
- » selects and aggregates records from the new table

Compile and Run the Sample Program

This section walks you through compiling and running the `SpliceODBCConnect.py` example program, in the following steps:

1. Install the Splice Machine ODBC driver

Follow our instructions for installing the driver on Unix or Windows.

2. Install the pyodbc module

You need to install the `pyodbc` open source Python module, which implements the DB API 2.0 specification and can be used with Python 2.4 or higher. See <https://github.com/mkleehammer/pyodbc> for more information about this module.

To install `pyodbc` on the server on which you'll be running your job:

```
yum install gcc-c++ pip install pyodbc
```

3. Confirm that you can connect

To confirm that you're ready to use the ODBC driver, launch the python shell and enter the following commands, replacing `SpliceODBC64` with the name of your data source (which is found in the `odbc.ini` file that you edited when installing our ODBC driver):

```
import pyodbc
cnxn = pyodbc.connect("DSN=SpliceODBC64")
cursor = cnxn.cursor()
cursor.execute("select * from SYS.SYSTABLES")
row = cursor.fetchone()
print('row:', row)
```

4. Copy the example program code:

You can copy and paste the code below:

```

#!/usr/bin/python

# This program is used to demonstrate connecting to Splice Machine using
ODBC

import pyodbc

#Connect to Splice Machine using an Datasource
cnxn = pyodbc.connect("DSN=SpliceODBC64")

#Open a cursor
cursor = cnxn.cursor()

#Build a select statement
cursor.execute("select * from SYS.SYSTABLES")

#Fetch one record from the select
row = cursor.fetchone()

#If there is a record, print it
if row:
    print(row)

#The following will continue to retrieve one record at a time from the re
sultset
while 1:
    row = cursor.fetchone()
    if not row:
        break
    print('table name:', row.TABLENAME)

#The following is an example of using the fetchall option, instead of ret
rieving one record at time
cursor.execute("select * from SYS.SYSSCHEMAS")
rows = cursor.fetchall()
for row in rows:
    print(row.SCHEMAID, row.SCHEMANAME)

#Create a table
cursor.execute("CREATE TABLE MYPYTHONTABLE(a int, b varchar(30))")

#Insert data into the table
cursor.execute("insert into MYPYTHONTABLE values (1,'a')");
cursor.execute("insert into MYPYTHONTABLE values (2,'b')");
cursor.execute("insert into MYPYTHONTABLE values (3,'c')");
cursor.execute("insert into MYPYTHONTABLE values (4,'c')");
cursor.execute("insert into MYPYTHONTABLE values (5,'c')");

```

```
#Commit the creation of the table  
cnxn.commit();  
  
#Confirm the records are in the table  
row = cursor.execute("select count(1) as TOTAL from SPLICE.MYPYTHONTABLE").fetchone()  
print(row.TOTAL)
```

5. Save the code to `SpliceODBCConnect.py`.

6. Run the program:

Run the `SpliceODBCConnect.py` program as follows:

```
python ./SpliceODBCConnect.py
```

Connecting to Splice Machine with C and ODBC

This topic shows you how to compile and run a sample C program that exercises the Splice Machine ODBC driver. The `listODBCdriver` program verifies that the driver is correctly installed and available.

Compile and Run the Sample Program

This section walks you through compiling and running the `listODBCdriver` example program, which simply displays information about the installed driver.

1. Install the ODBC driver

Follow our instructions for installing the driver on Unix or Windows.

2. Copy the example program code

You can copy and paste the code below:

```
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>

main() {
    SQLHENV env;
    char driver[256];
    char attr[256];
    SQLSMALLINT driver_ret;
    SQLSMALLINT attr_ret;
    SQLUSMALLINT direction;
    SQLRETURN ret;

    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
    SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION, (void *) SQL_OV_ODBC3, 0);

    direction = SQL_FETCH_FIRST;
    while(SQL_SUCCEEDED(ret = SQLDrivers(env, direction,
        driver, sizeof(driver), &driver_ret,
        attr, sizeof(attr), &attr_ret))) {
        direction = SQL_FETCH_NEXT;
        printf("%s - %s\n", driver, attr);
        if (ret == SQL_SUCCESS_WITH_INFO) printf("\tdata truncation\n");
    }
}
```

3. Compile it

```
#!/bin/bash
# gcc -I /usr/local/splice/unixODBC/include listODBCdriver.c -o listODBCd
river -L/usr/local/splice/lib -lodb -lodbinst -lodbccr
```

4. Run the program

Run the compiled listODBCdriver:

```
prompt:~$ ./listODBCdriver
```

The command should display a result like the following:

```
Splice Machine - Description=Splice Machine ODBC Driver
```

Ingesting and Streaming Data With Splice Machine

This section provides tutorials to help you stream data with Splice Machine, in these topics:

- » Using Apache Storm
- » MQTT Spark Streaming
- » Creating a Kafka Producer
- » Configuring a Kafka Feed

Integrating Apache Storm with Splice Machine

This topic walks you through building and running two different examples of integrating Storm with Splice Machine:

- » [Inserting Random Values in Splice Machine with Storm](#)
- » [Inserting Data into Splice Machine from MySQL](#)

Inserting Random Values in Splice Machine with Storm

This example iterates through a list of random words and numbers, inserting the values into a Splice Machine database; follow along in these steps:

1. Download the Sample Code:

Pull the code from our git repository:

```
https://github.com/splicemachine/splice-community-sample-code/tree/master/tutorial-storm
```

2. Check the prerequisites:

You must be running Splice Machine 2.0 or later on the same machine on which you will run this example code. If Splice Machine is running on a different machine, you'll need to modify the `server` variable in the `SpliceDumperTopology.java` file; change it to the name of the server that is running Splice Machine.

You also must have `maven v.3.3.9` or later installed.

3. Create the test table in Splice Machine:

This example inserts data into a Splice Machine table named `testTable`. You need to create that table by entering this command at the `splice>` prompt:

```
splice> CREATE TABLE testTable( word VARCHAR(100), number INT );
```

4. Compile the sample code:

Compile the sample code with this command

```
% mvn clean compile dependency:copy-dependencies
```

5. Run the sample code:

Follow these steps:

- a. Make sure that Splice Machine is running and that you have created the `testTable` table.
- b. Execute this script to run the program:

```
% run-storm.sh
```

- c. Query `testTable` in Splice Machine to verify that it has been populated with random words and numbers:

```
splice> select * from testTable;
```

About the Sample Code Classes

The random insertion example contains the following java classes, each of which is described below:

Class	Description
<code>SpliceCommunicator.java</code>	Contains methods for communicating with Splice Machine.
<code>SpliceConnector.java</code>	Establishes a JDBC connection with Splice Machine.
<code>SpliceDumperBolt.java</code>	Dumps data into Splice Machine.
<code>SpliceDumperTopology.java</code>	Defines the Storm topology for this example.
<code>SpliceIntegerSpout.java</code>	Emits tuples that are inserted into the Splice Machine table.

Inserting Data into Splice Machine from MySQL

This example uses Storm to read data from a MySQL database, and insert that data into a table in Splice Machine.

1. Download the Sample Code:

Pull the code from our git repository:

```
https://github.com/splicemachine/splice-community-sample-code/tree/master/tutorial-storm
```

2. Check the prerequisites:

You must be running Splice Machine 2.0 or later on the same machine on which you will run this example code. If Splice Machine is running on a different machine, you'll need to modify the `server` variable in the `MySqlToSpliceTopology.java` file; change it to the name of the server that is running Splice Machine.

You also must have `maven v.3.3.9` or later installed.

This example assumes that your MySQL database instance is running on the same machine on which you're running Splice Machine, and that the root user does not have a password. If either of these is not true, then you need to modify the call to the `seedBufferQueue` method in the `MySqlSpout.java` file. This method takes four parameters that you may need to change:

```
seedBufferQueue( MySqlServer, MySqlDatabase, mySqlUserName, mySqlPassword );
```

The default settings used in this example are:

```
seedBufferQueue( "localhost", "test", "root", "" );
```

3. Create the students table in Splice Machine:

This example inserts data into a Splice Machine table named `students`. You need to create that table by entering this command at the `splice>` prompt:

```
splice> CREATE TABLE students( name VARCHAR(100) );
```

4. Create the students table in your MySQL database:

This example read data from a MySQL table named `students`. You need to create that table in MySQL:

```
$$ CREATE TABLE students( id INTEGER, name VARCHAR(100) );
```

If your MySQL instance is on a different machine

5. Compile the sample code:

Compile the sample code with this command

```
% mvn clean compile dependency:copy-dependencies
```

6. Run the sample code:

Follow these steps:

a. Make sure that Splice Machine is running and that you have created the `testTable` table.

b. Execute this script to run the program:

```
% run-mysql-storm.sh
```

c. Query the `students` table in Splice Machine to verify that it has been populated with data from the MySQL table:

```
splice> select * from students;
```

About the Sample Code Classes

This example contains the following java classes:

Class	Description
<code>MySqlCommunicator.java</code>	Contains methods for communicating with MySQL.
<code> MySqlConnector.java</code>	Establishes a JDBC connection with MySQL.
<code> MySqlSpliceBolt.java</code>	Dumps data from MySQL into Splice Machine.
<code> MySqlSpout.java</code>	Emits tuples from MySQL that are inserted into the Splice Machine table.
<code> MySqlToSpliceTopology.java</code>	Defines the Storm topology for this example.
<code> SpliceCommunicator.java</code>	Contains methods for communicating with Splice Machine.
<code> SpliceConnector.java</code>	Establishes a JDBC connection with Splice Machine.

Streaming MQTT Spark Data

This topic walks you through using MQTT Spark streaming with Splice Machine. MQTT is a lightweight, publish-subscribe messaging protocol designed for connecting remotely when a small footprint is required. MQTT is frequently used for data collection with the Internet of Things (IoT).

The example code in this tutorial uses [Mosquitto](#), which is an open source message broker that implements the MQTT. This tutorial uses a cluster managed by MapR; if you're using different platform management software, you'll need to make a few adjustments in how the code is deployed on your cluster.

NOTE: All of the code used in this tutorial is available in our [GitHub community repository](#).

You can complete this tutorial by [watching a short video](#) or by [following the written directions](#) below.

Watch the Video

The following video shows you how to:

- » put messages on an MQTT queue
- » consume those messages using Spark streaming
- » save those messages to Splice Machine with a virtual table (VTI)

Written Walk Through

This section walks you through the same sequence of steps as the video, in these sections:

- » [Deploying the Tutorial Code](#) walks you through downloading and deploying the sample code.
- » [About the Sample Code](#) describes the high-level methods in each class.
- » [About the Sample Code Scripts](#) describes the scripts used to deploy and execute the sample code.

Deploying the Tutorial Code

Follow these steps to deploy the tutorial code:

1. **Download the code from our [GitHub community repository](#).**

Pull the code from our git repository:

<https://github.com/splicemachine/splice-community-sample-code/tree/master/tutorial-mqtt-spark-streaming>

2. Compile and package the code:

```
mvn clean compile package
```

3. Copy three JAR files to each server:

Copy these three files:

```
./target/splice-tutorial-mqtt-2.0.jarspark-streaming-mqtt_2.10-1.6.1.jar  
org.eclipse.paho.client.mqttv3-1.1.0.jar
```

to this directory on each server:

```
/opt/splice/default/lib
```

4. Restart Hbase

5. Create the target table in splice machine:

Run this script to create the table:

```
create-tables.sql
```

6. Start Mosquitto:

```
sudo su /usr/sbin/mosquitto -d -c /etc/mosquitto/mosquitto.conf > /var/log/mosquitto.log 2>&1
```

7. Start the Spark streaming script:

```
sudo -su mapr ./run-mqtt-spark-streaming.sh tcp://srv61:1883 /testing 10
```

The first parameter (`tcp://srv61:1883`) is the MQTT broker, the second (`/testing`) is the topic name, and the third (10) is the number of seconds each stream should run.

8. Start putting messages on the queue:

Here's a java program that is set up to put messages on the queue:

```
java -cp /opt/splice/default/lib/splice-tutorial-mqtt-2.0-SNAPSHOT.jar:/opt/splice/default/lib/org.eclipse.paho.client.mqttv3-1.1.0.jar com.splice.machine.tutorials.sparkstreaming.mqtt.MQTTPublisher tcp://localhost:1883 /testing 1000 R1
```

The first parameter (`tcp://localhost:1883`) is the MQTT broker, the second (`/testing`) is the topic name, the third (1000) is the number of iterations to execute, and the fourth parameter (R1) is a prefix for this run.

NOTE: The source code for this utility program is in a different GitHub project than the rest of this code. You'll find it in the [tutorial-kafka-producer](#) Github project.

About the Sample Code

This section describes the main class methods used in this MQTT example code; here's a summary of the classes:

Java Class	Description
MQTTPublisher	Puts csv messages on an MQTT queue.
SparkStreamingMQTT	The Spark streaming job that reads messages from the MQTT queue.
SaveRDD	Inserts the data into Splice Machine using the RFIDMessageVTI class.
RFIDMessageVTI	A virtual table interface for parsing an RFIDMessage.
RFIDMessage	Java object (a POJO) for converting from a csv string to an object to a database entry.

[MQTTPublisher](#)

This class puts CSV messages on an MQTT queue. The function of most interest in `MQTTPublisher.java` is `DoDemo`, which controls our sample program:

```

public void doDemo() {
    try {
        long startTime = System.currentTimeMillis();
        client = new MqttClient(broker, clientId);
        client.connect();
        MqttMessage message = new MqttMessage();
        for (int i=0; i<numMessages; i++) {
            // Build a csv string
            message.setPayload( prefix + "Asset" + i + ", Location" + i + "," + new Ti
mestamp((new Date()).getTime()).getBytes());
            client.publish(topicName, message);
            if (i % 1000 == 0)
                System.out.println("records:" + i + " duration=" + (System.currentTimeMillis()
- startTime));
            startTime = System.currentTimeMillis();
        }
        client.disconnect();
    } catch (MqttException e) {
        e.printStackTrace();
    }
}

```

DoDemo does a little initialization, then starts putting messages out on the queue. Our sample program is set up to loop until it creates numMessages messages; after every 1000 messages, it displays a status message that helps us determine how much time is going to put messages on the queue, and how much to take them off the queue.

DoDemo builds a csv record (line) for each message, setting an asset ID, a location ID, and a timestamp in the payload of the message. It then publishes that message to the topic topicName.

SparkStreamingMQTT

Once the messages are on the queue, our SparkStreamingMQTT class object reads them from the queue and inserts them into our database. The main method in this class is processMQTT:

```

public void processMQTT(final String broker, final String topic, final int numSeconds) {
    LOG.info("***** SparkStreamingMQTTOutside.processMQTT start");

    // Create the spark application and set the name to MQTT
    SparkConf sparkConf = new SparkConf().setAppName("MQTT");

    // Create the spark streaming context with a 'numSeconds' second batch size
    jssc = new JavaStreamingContext(sparkConf, Durations.seconds(numSeconds));
    jssc.checkpoint(checkpointDirectory);

    LOG.info("***** SparkStreamingMQTTOutside.processMQTT about to read the MQTTUtils.createStream");
    //2. MQTTUtils to collect MQTT messages
    JavaReceiverInputDStream<String> messages = MQTTUtils.createStream(jssc, broker, topic);

    LOG.info("***** SparkStreamingMQTTOutside.processMQTT about to do foreach RDD");
    //process the messages on the queue and save them to the database
    messages.foreachRDD(new SaveRDD());

    LOG.info("***** SparkStreamingMQTTOutside.processMQTT prior to context.start");
    // Start the context
    jssc.start();
    jssc.awaitTermination();
}

```

The `processMQTT` method takes three parameters:

broker

The URL of the MQTT broker.

topic

The MQTT topic name.

numSeconds

The number of seconds at which streaming data will be divided into batches.

The `processMQTT` method processes the messages on the queue and saves them by calling the `SaveMDD` class.

SaveRDD

The `SaveRDD` class is an example of a Spark streaming function that uses our virtual table interface (VTI) to insert data into your Splice Machine database. This function checks for messages in the stream, and if there any, it creates a connection your database and uses a prepared statement to insert the messages into the database.

```

/**
 * This is an example of spark streaming function that
 * inserts data into Splice Machine using a VTI.
 *
 * @author Erin Driggers
 */

public class SaveRDD implements FunctionJavaRDDString>, Void>, Externalizable {

    private static final Logger LOG = Logger.getLogger(SaveRDD.class);

    @Override
    public Void call(JavaRDDString> rddRFIDMessages) throws Exception {
        LOG.debug("About to read results:");
        if (rddRFIDMessages != null & rddRFIDMessages.count() > 0) {
            LOG.debug("Data to process:");
            //Convert to list
            ListString> rfidMessages = rddRFIDMessages.collect();
            int numRcds = rfidMessages.size();

            if (numRcds > 0) {
                try {
                    Connection con = DriverManager.getConnection("jdbc:splice://localhost:1527/splicedb;user=splice;password=admin");

                    //Syntax for using a class instance in a VTI, this could also be a table function
                    String vtiStatement = "INSERT INTO IOT.RFID "
                        + "select s.* from new com.splicemachine.tutorials.spark.streaming.mqtt.RFIDMessageVTI(?) s ("
                        + RFIDMessage.getTableDefinition() + ")";
                    PreparedStatement ps = con.prepareStatement(vtiStatement);
                    ps.setObject(1, rfidMessages);
                    ps.execute();
                } catch (Exception e) {
                    //It is important to catch the exceptions as log messages because it is difficult
                    //to trace what is happening otherwise
                    LOG.error("Exception saving MQTT records to the database" + e.getMessage(), e);
                } finally {
                    LOG.info("Complete insert into IOT.RFID");
                }
            }
        }
        return null;
    }
}

```

The heart of this function is the statement that creates the prepared statement, using a VTI class instance:

```

String vtiStatement = "INSERT INTO IOT.RFID " + "select s.* from new com.splicemachine.tutorials.sparkstreaming.mqtt.RFIDMessageVTI() s (" +
    + RFIDMessage.getTableDefinition() + ")";
PreparedStatement ps = con.prepareStatement(vtiStatement);

```

Note that the statement references both our `RFIDMessage` and `RFIDMessageVTI` classes, which are described below.

RFIDMessageVTI

The `RFIDMessageVTI` class implements an example of a virtual table interface that reads in a list of strings that are in CSV format, converts that into an `RFIDMessage` object, and returns the resultant list in a format that is compatible with Splice Machine.

This class features an override of the `getDataSet` method, which loops through each CSV record from the input stream and converts it into an `RFIDMessage` object that is added onto a list of message items:

```

@Override
public DataSetLocatedRow> getDataSet(SpliceOperation op, DataSetProcessor dsp, ExecRow execRow) throws StandardException {
    operationContext = dsp.createOperationContext(op);

    //Create an arraylist to store the key / value pairs
    ArrayListLocatedRow> items = new ArrayListLocatedRow>();

    try {

        int numRcds = this.records == null ? 0 : this.records.size();

        if (numRcds > 0) {

            LOG.info("Records to process:" + numRcds);
            //Loop through each record convert to a SensorObject
            //and then set the values
            for (String csvString : records) {
                CsvBeanReader beanReader = new CsvBeanReader(new StringReader(csvString), CsvPreference.STANDARD_PREFERENCE);
                RFIDMessage msg = beanReader.read(RFIDMessage.class, header, processors);
                items.add(new LocatedRow(msg.getRow()));
            }
        }
    } catch (Exception e) {
        LOG.error("Exception processing RFIDMessageVTI", e);
    } finally {
        operationContext.popScope();
    }
    return new ControlDataSet>(items);
}

```



For more information about using our virtual table interface, see [Using the Splice Machine Virtual Table Interface](#).

RFIDMessage

The `RFIDMessage` class creates a simple Java object (a POJO) that represents an RFID message; we use this to convert an incoming CSV-formatted message into an object. This class includes getters and setters for each of the object properties, plus the `getTableDefinition` and `getRow` methods:

```
/**
 * Used by the VTI to build a Splice Machine compatible resultset
 *
 * @return
 * @throws SQLException
 * @throws StandardException
 */
public ValueRow getRow() throws SQLException, StandardException {
    ValueRow valueRow = new ValueRow(5);
    valueRow.setColumn(1, new SQLVarchar(this.getAssetNumber()));
    valueRow.setColumn(2, new SQLVarchar(this.getAssetDescription()));
    valueRow.setColumn(3, new SQLTimestamp(this.getRecordedTime()));
    valueRow.setColumn(4, new SQLVarchar(this.getAssetType()));
    valueRow.setColumn(5, new SQLVarchar(this.getAssetLocation()));
    return valueRow;
}

/**
 * Table definition to use when using a VTI that is an instance of a class
 *
 * @return
 */
public static String getTableDefinition() {
    return "ASSET_NUMBER varchar(50), "
        + "ASSET_DESCRIPTION varchar(100), "
        + "RECORDED_TIME TIMESTAMP, "
        + "ASSET_TYPE VARCHAR(50), "
        + "ASSET_LOCATION VARCHAR(50) ";
}
```

The `getTableDefinition` method is a string description of the table into which you're inserting records; this pretty much replicates the specification you would use in an SQL `CREATE_TABLE` statement.

The `getRow` method creates a data row with the appropriate number of columns, uses property getters to set the value of each column, and returns the row as a `resultset` that is compatible with Splice Machine.

About the Sample Code Scripts

These are also two scripts that we use with this tutorial:

Class	Description
/ddl/create-tables.sql	A simple SQL script that you can use to have Splice Machine create the table into which RFID messages are stored.
/scripts/run-mqtt-spark-streaming.sh	Starts the Spark streaming job.

Streaming Data with Kafka: Creating a Producer

This topic demonstrates how to create a Kafka Producer to feed data into Splice Machine; we'll subsequently use this producer in other tutorials.

Watch the Video:

The following video shows you how to create a Kafka producer to feed data into Splice Machine.

Streaming Data with Kafka: Configuring a Feed

This topic demonstrates how to create a Kafka Feed , which puts messages on a Kafka Queue. We'll make use of this class in other tutorials.

Watch the Video:

The following video shows you how to configure a Kafka feed to Splice Machine.

Connecting Splice Machine with Business Intelligence Tools

This section shows you how to connect specific Business Intelligence tools to your Splice Machine database, including:

- » Connecting Cognos to Splice Machine
- » Connecting DBeaver to Splice Machine
- » Connecting DBVisualizer to Splice Machine
- » Connecting SQuirreL to Splice Machine
- » Connecting Tableau to Splice Machine

Connecting Cognos with Splice Machine Using ODBC

This topic shows you how to connect Cognos to Splice Machine using our ODBC driver. To complete this tutorial, you need to:

- » Have Cognos installed on your Windows or MacOS computer. You can find directions on the IBM web site (<http://www-03.ibm.com/software/products/en/cognos-analytics>); you can also download a free trial version of Cognos from there.
- » Have the Splice Machine ODBC driver installed on your computer. Follow our instructions.

Watch the Video

The following video shows you how to connect Cognos to Splice Machine using our ODBC driver.

Connecting DBeaver with Splice Machine Using JDBC

This topic shows you how to connect DBeaver to Splice Machine using our JDBC driver. To complete this tutorial, you need to:

- » Have Splice Machine installed and running on your computer.
- » Have DBeaver installed on your computer. You can download an installer and find directions on the DBeaver web site (dbeaver.jkiss.org).

Connect DBeaver with Splice Machine

This section walks you through configuring DBeaver to connect with Splice Machine

1. Install DBeaver , if you've not already done so

[Follow the instructions on the DBeaver web site.](#)

2. Start a Splice Machine session on the computer on which you have installed DBeaver

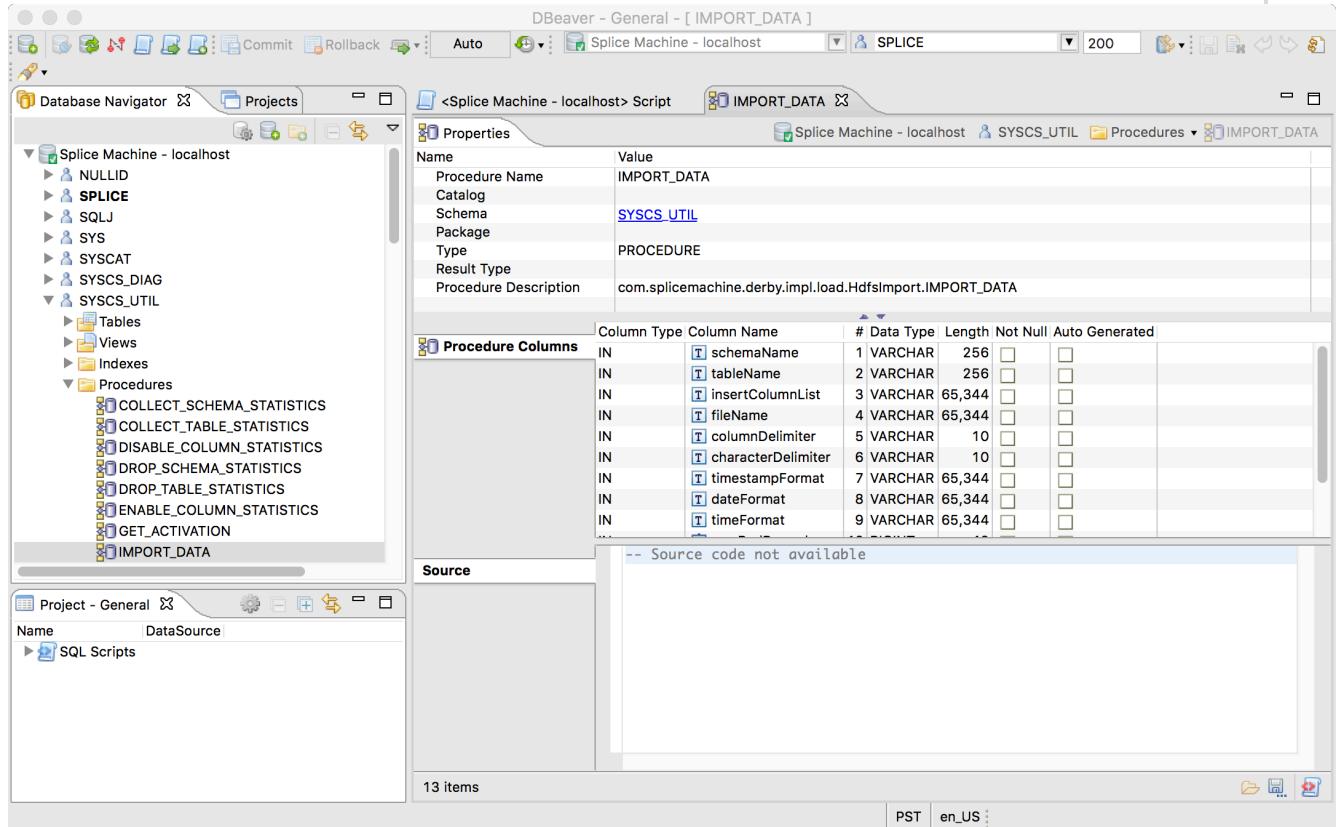
Splice Machine must be running to create and use it with DBeaver .

3. Configure a Splice Machine connection in DBeaver

Follow the instructions in the next section, [Configure a DBeaver Connection for Splice Machine](#), to create and test a new connection in DBeaver .

4. Connect DBeaver to Splice Machine

In DBeaver's *Database Navigator*, select the Splice Machine connection you configured. Your database will display, and you can inspect objects or enter SQL to interact with your data.



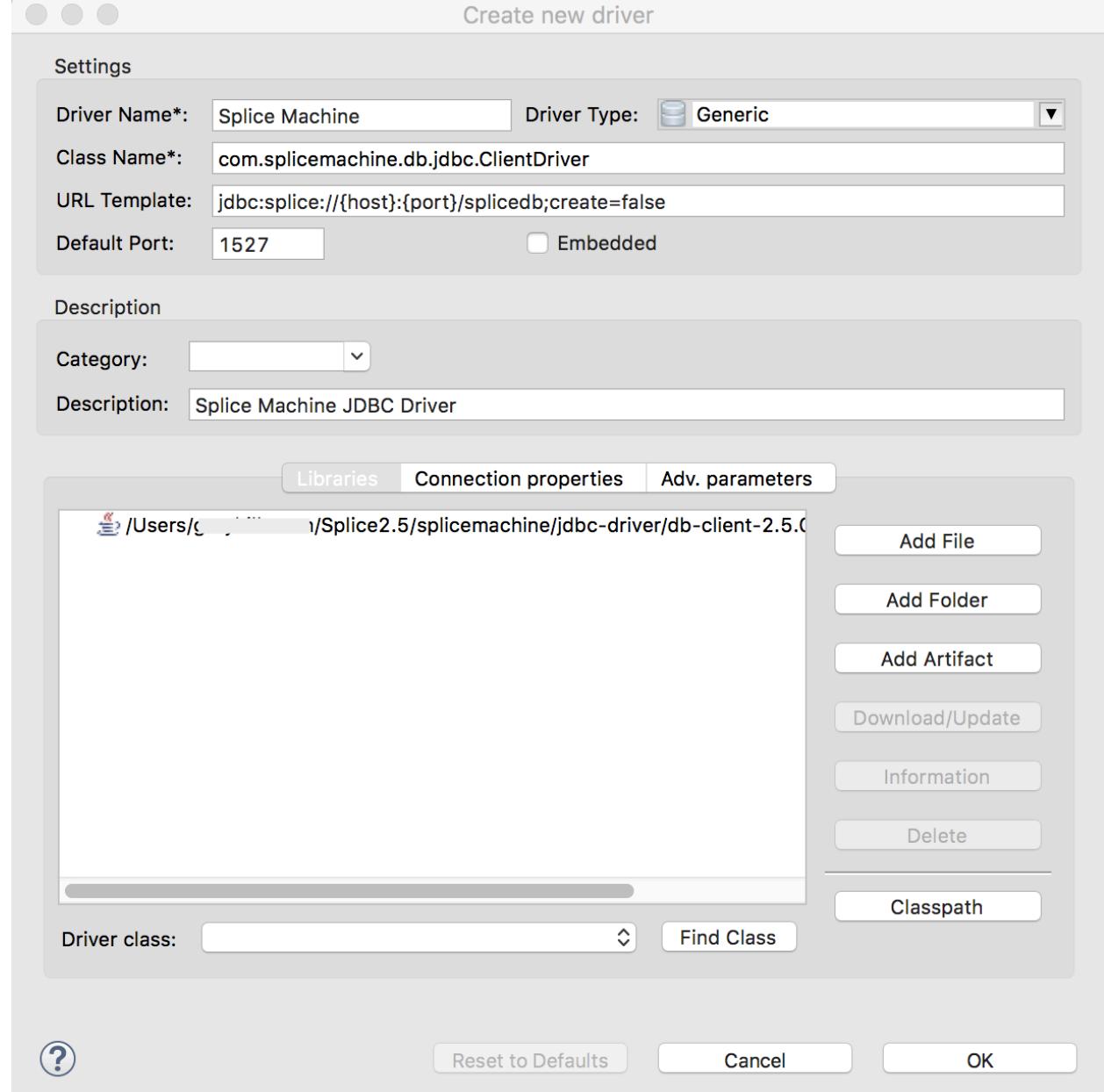
Configure a DBeaver Connection for Splice Machine

Follow these steps to configure and test a new driver and connection alias in DBeaver .

1. Start a Splice Machine session on the computer on which you have installed DBeaver
2. Open the DBeaver application.
3. Select **Driver Manager** in the DBeaver *Database* menu, then click the **New** button to create a new driver:
 - a. Specify values in the *Create New Driver* form; these are the default values:

Field	Value
Driver Name:	Any name you choose
Class Name:	com.splicemachine.db.jdbc.ClientDriver
URL Template:	jdbc:splice://{{host}}:{{port}}/splicedb;create=false
Default Port:	1527
Description:	Any description you want to specify

- b. Click the Add File button, then navigate to and select the Splice JDBC Driver jar file. which you'll find it in the jdbc-driver folder under the splicemachine directory on your computer.

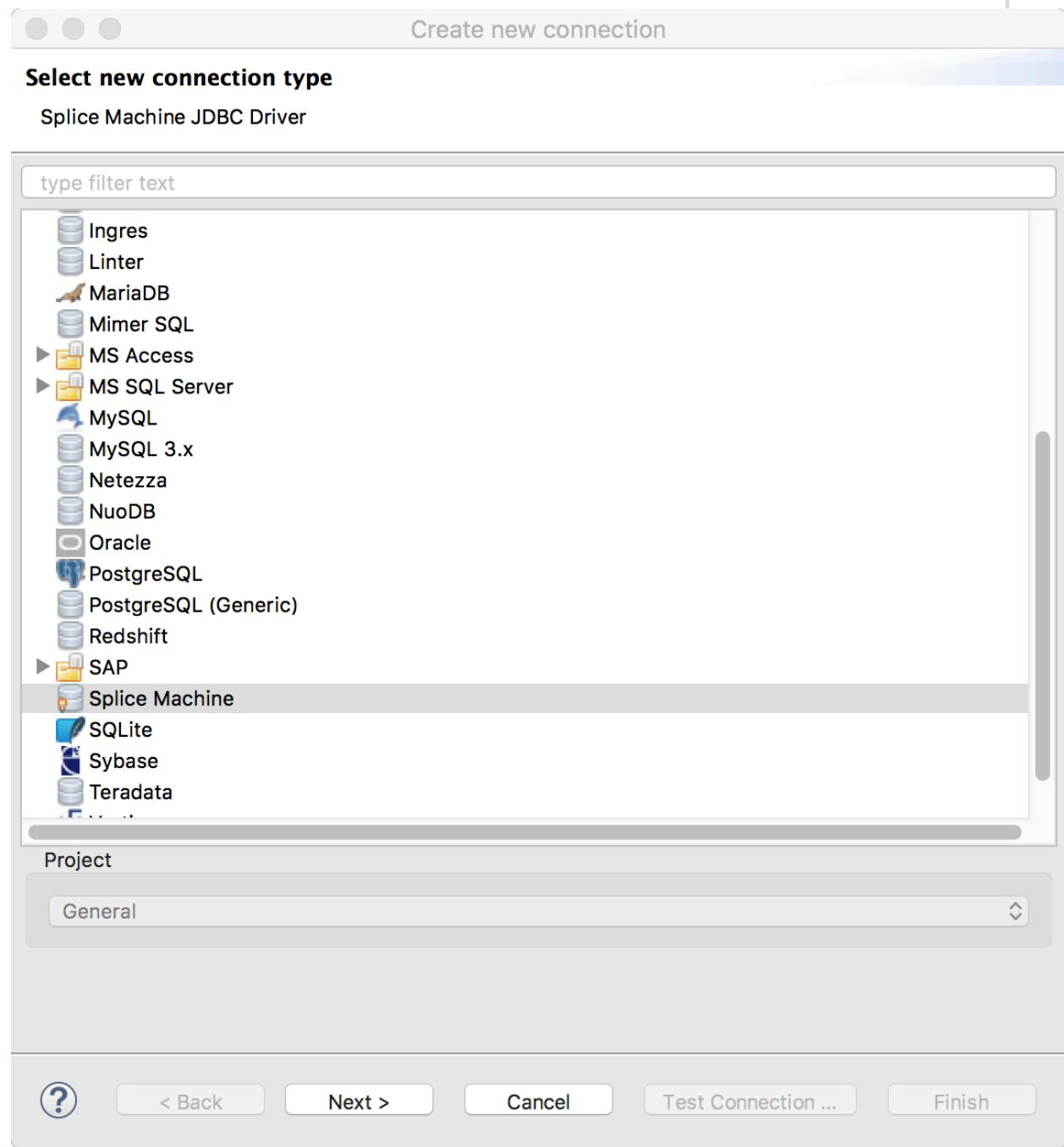


NOTE: Instead of manually entering the Class Name for your driver, you can click the Find Class button to discover the driver class name associated with the file you've located.

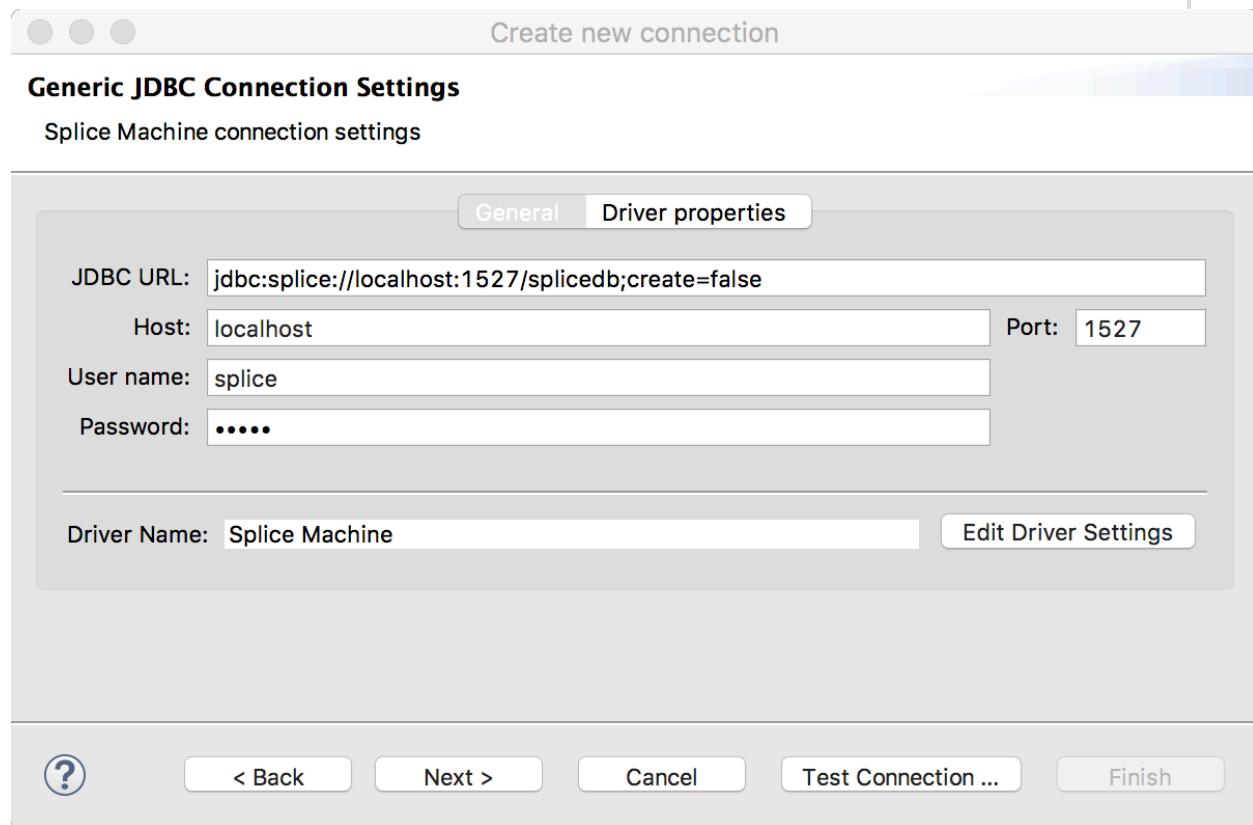
4. Click OK to save the driver entry and close the form.

5. Select **New Connection** in the DBeaver Database menu, then follow these steps to create a new connection that uses our driver:

- a. Scroll through the connection type list and select the Splice Machine JDBC driver that you just created, then click the **Next** button:



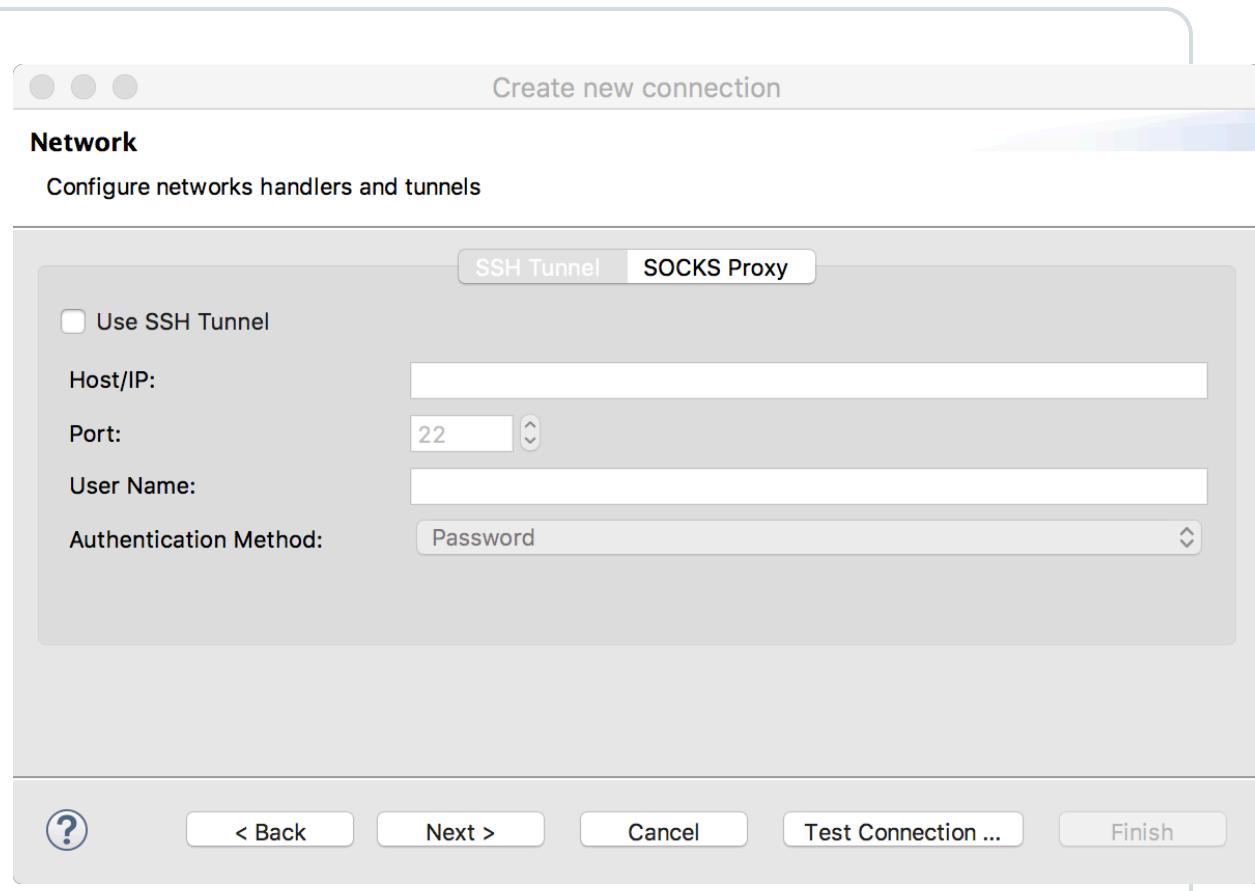
- b. Several of the fields in the *Generic JDBC Connection Settings* screen were pre-populated for you when you selected the driver. You need to fill in the **User name:** (default is `splice`) and **Password:** (default is `admin`) field values:



- c. Click the **Test Connection** button to verify your connection.

NOTE: Splice Machine must be running on your computer for the connection test to succeed.

- d. Click the **Next** button to reveal the network configuration screen. If you have VPN requirements, enter the appropriate information in this screen; if not, simply click the **Next** button again.



- e. You can optionally modify any settings in the *Finish connection creation* screen; then click the **Finish** button to save your new connection.

Create new connection

Finish connection creation

General connection settings.

Connection name: Splice Machine - localhost

Connection type: Development ▾ Edit

Connection folder: <None> ▾

Security

Save password locally

Miscellaneous

Show system objects

Show utility objects

Connection

Auto-commit:

Isolation level:

Default schema:

Keep-Alive: 0 ▾

?

< Back

Next >

Cancel

Test Connection ...

Finish

Connecting DBVisualizer with Splice Machine Using JDBC

This topic shows you how to connect DBVisualizer to Splice Machine using our JDBC driver. To complete this tutorial, you need to:

- » Have Splice Machine installed and running on your computer.
- » Have DBVisualizer installed on your computer. You can find directions on the DBVisualizer web site (<https://www.dbvis.com>); you can also download a free trial version of DBVisualizer from there.

Connect DBVisualizer with Splice Machine

This section walks you through configuring DBVisualizer to connect with Splice Machine

1. **Install DBVisualizer, if you've not already done so**

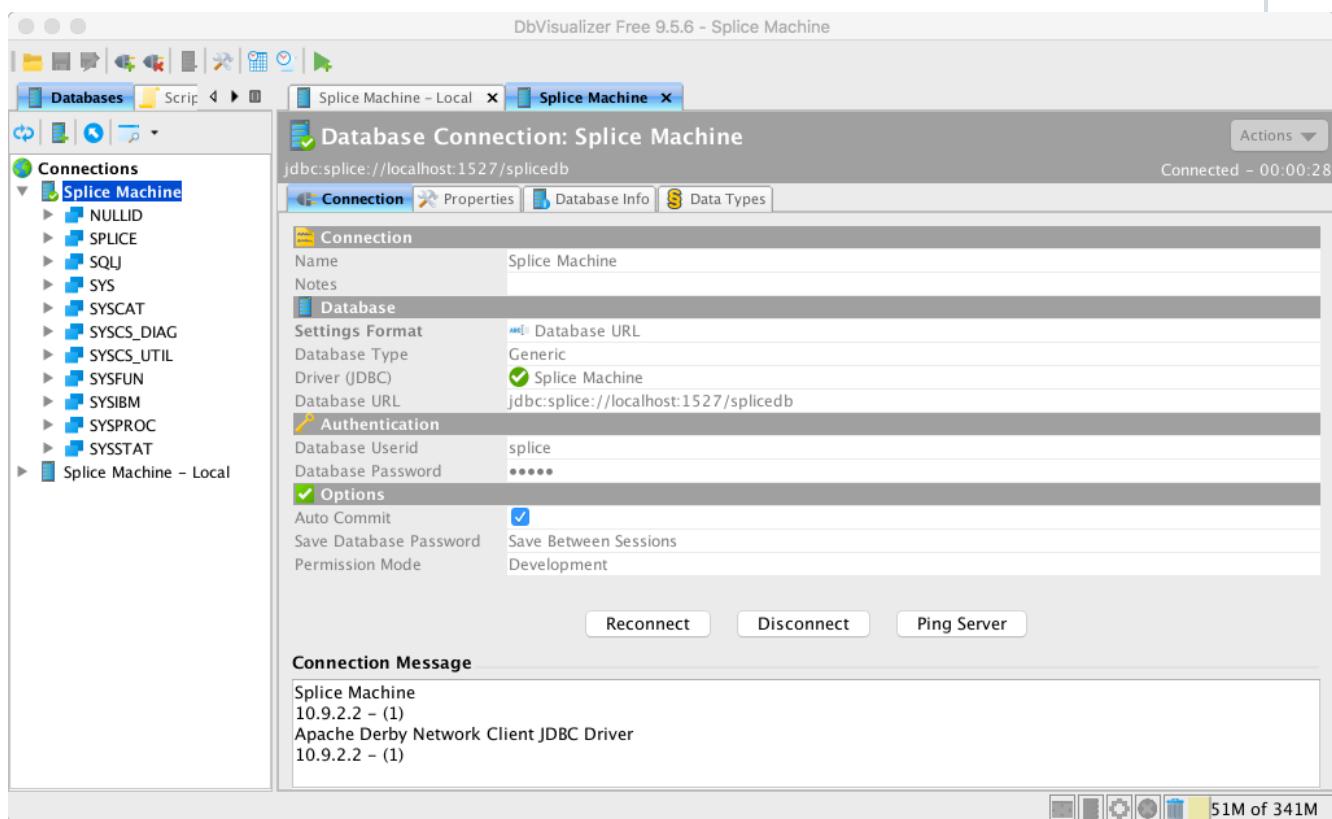
[Follow the instructions on the DBVis web site.](#)

2. **Configure a Splice Machine connection in DBVisualizer**

Follow the instructions in the next section, [Configure a DBVisualizer Connection for Splice Machine](#), to create and test a new connection in DBVisualizer.

3. **Connect DBVisualizer to Splice Machine**

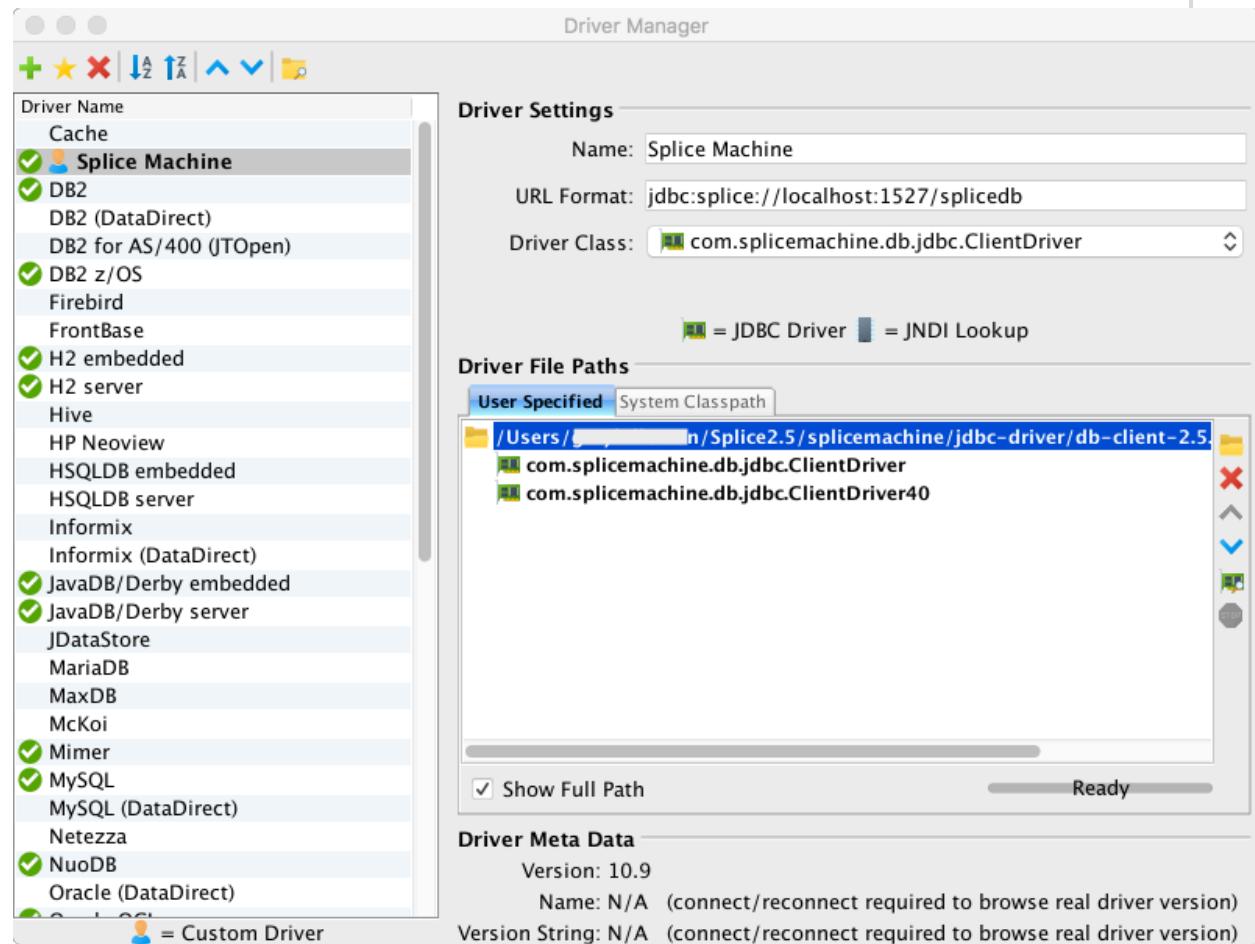
In DBVisualizer, open the connection alias you created and click the Connect button. Your database will display in DBVisualizer, and you can inspect objects or enter SQL to interact with your data.



Configure a DBVisualizer Connection for Splice Machine

Follow these steps to configure and test a new driver entry and connection in DBVisualizer.

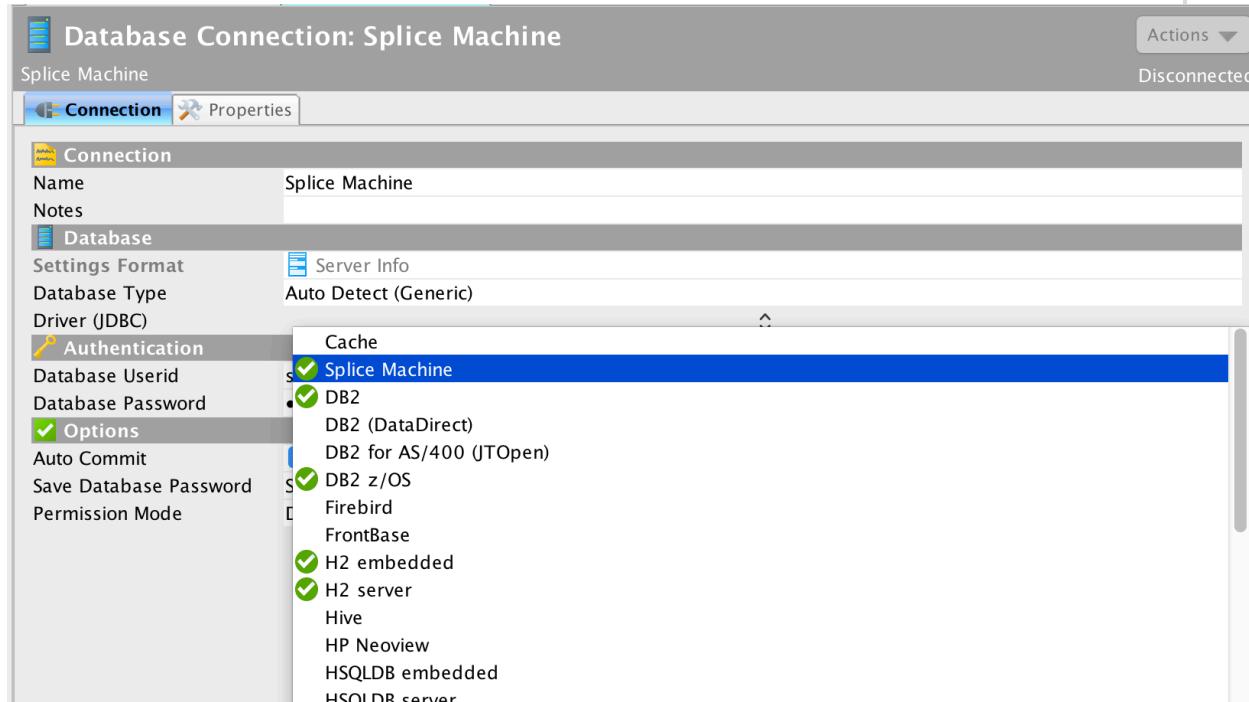
1. Start a Splice Machine session on the computer on which you have installed DBVisualizer.
2. Open the DBVisualizer application.
3. **Use the Driver Manager to create a new DBVisualizer driver entry.**
Select **Driver Manager** from the *Tools* menu; in the **Driver Manager** screen:
 - a. Click the green plus sign + button to add a new driver entry.
 - b. Name the driver and enter `jdbc:splice://localhost:1527/splicedb` in the **URL Format** field:



- c. In the Driver File Paths section, click User Specified, and then click the yellow folder icon.
- d. Navigate to and select the Splice JDBC Driver jar file, which you'll find it in the `jdbc-driver` folder under the `splicemachine` directory on your computer.
- e. Close the Driver Manager screen.

4. Create a DBVisualizer connection alias that uses the new driver:

- a. Select Create Database Connection from the Database menu. If prompted about using the Wizard, click the No Wizard button.
- b. Name the connection (we use Splice Machine), then click the empty field next to the Driver (JDBC) caption and select the driver you just created:



- Enter the following URL into the Database URL field that appears once you've selected the driver:

```
jdbc:splice://localhost:1527/spicedb
```



Use `localhost:1527` with the standalone (local computer) version of splicemachine. If you're running Splice Machine on a cluster, substitute the address of your server for `localhost`; for example:
`jdbc:splice://mySrv123cba:1527/spicedb.`

- Fill in the `Userid (splice)` and `Password (admin)` fields. Then click the Connect button. Your Splice Machine database will now display in DBVisualizer:

Connecting SQuirreL with Splice Machine Using JDBC

This topic shows you how to connect SQuirreL to Splice Machine using our JDBC driver. To complete this tutorial, you need to:

- » Have Splice Machine installed and running on your computer.
- » Have SQuirreL installed on your computer. You can find directions on the SQuirreL web site (<http://squirrel-sql.sourceforge.net/>); you can also download a free trial version of SQuirreL from there. You must also install the Derby plug-in for SQuirreL.

Connect SQuirreL with Splice Machine

This section walks you through configuring SQuirreL to connect with Splice Machine

1. **Install SQuirreL, if you've not already done so:**

[Follow the instructions on the SQuirreL web site.](#)

2. **Install the Derby plug-in for SQuirreL**

This plug-in is required to operate with Splice Machine. If you didn't select the Derby plug-in when you installed SQuirreL, you can [download Apache Derby here](#) and drop the plugin file into the plugin/ directory of your SQuirrel SQL installation directory. See [SQuirrel's Plugin Overview](#) for more info.

3. **Start a Splice Machine session on the computer on which you have installed SQuirreL**

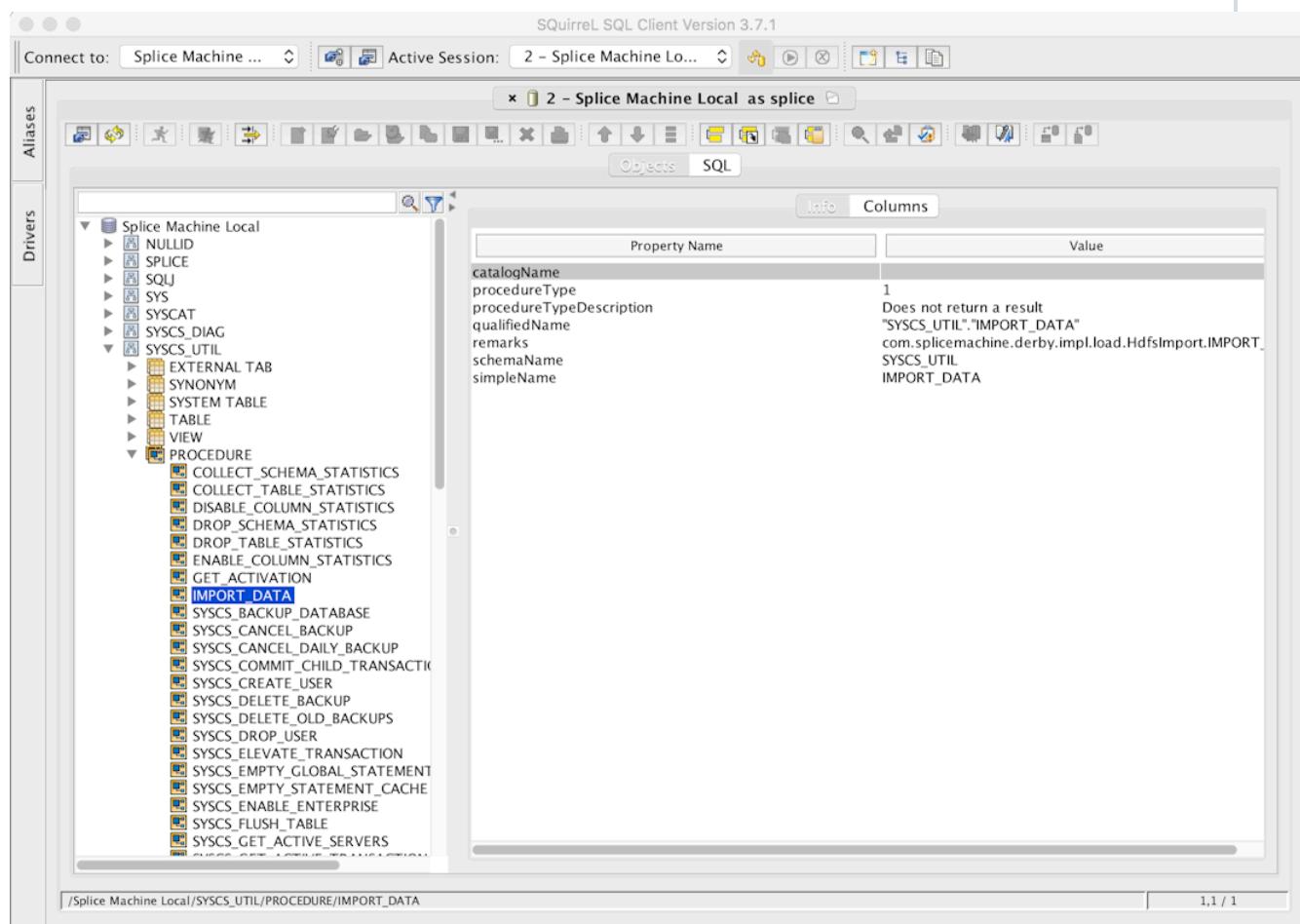
Splice Machine must be running to create and use it with SQuirreL.

4. **Configure a Splice Machine connection in SQuirreL**

Follow the instructions in the next section, [Configure a SQuirreL Connection for Splice Machine](#), to create and test a new connection in SQuirreL.

5. **Connect SQuirreL to Splice Machine**

In SQuirreL, open the connection alias you created, enter your credentials, and click the Connect button. Your database will display in SQuirreL, and you can inspect objects or enter SQL to interact with your data.



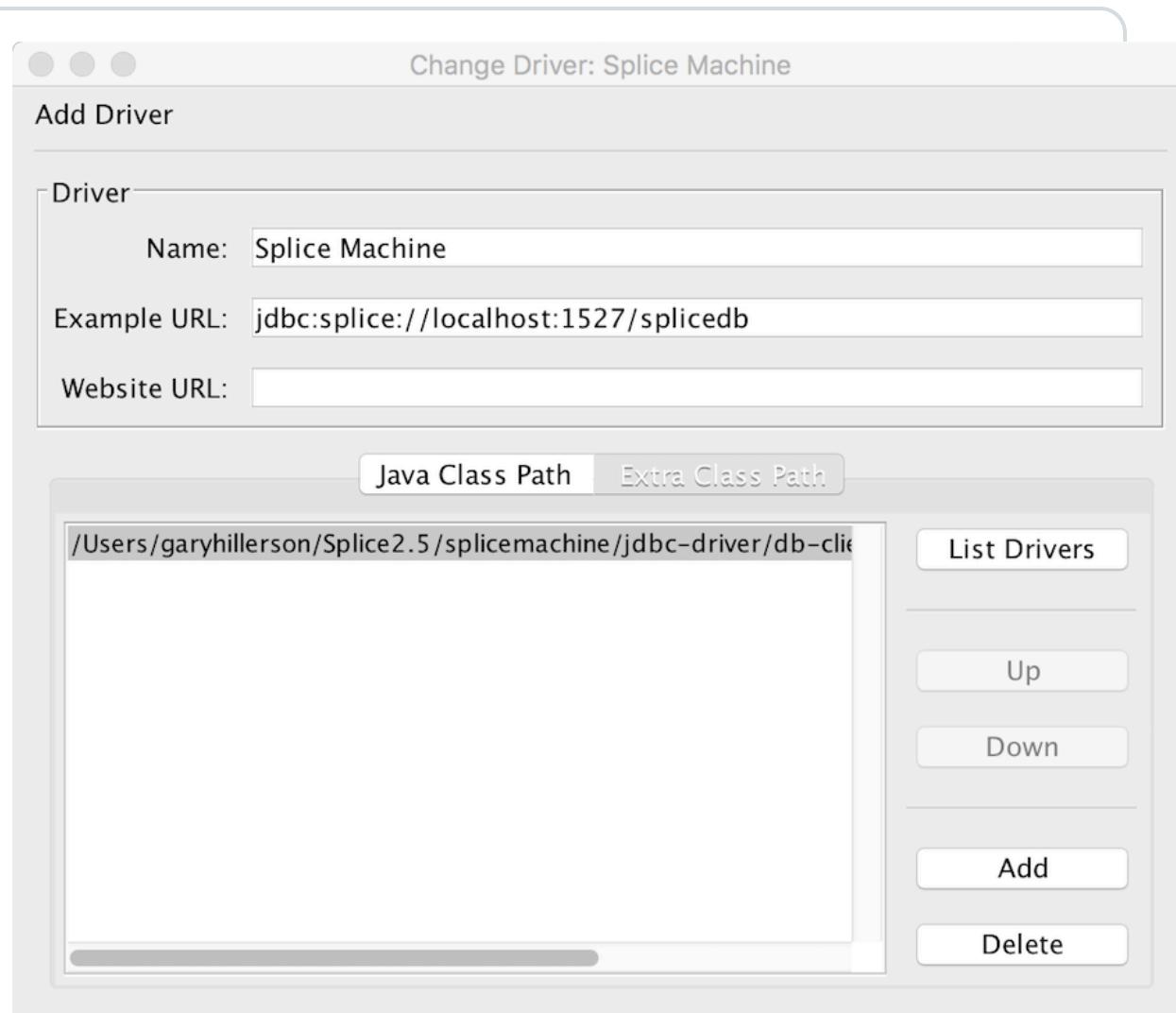
Configure a SQuirreL Connection for Splice Machine

Follow these steps to configure and test a new driver and connection alias in SQuirreL.

1. Start a Splice Machine session on the computer on which you have installed SQuirreL
2. Open the SQuirreL application.
3. Click the **SQuirreL Drivers** tab, which is near the upper left of the window:



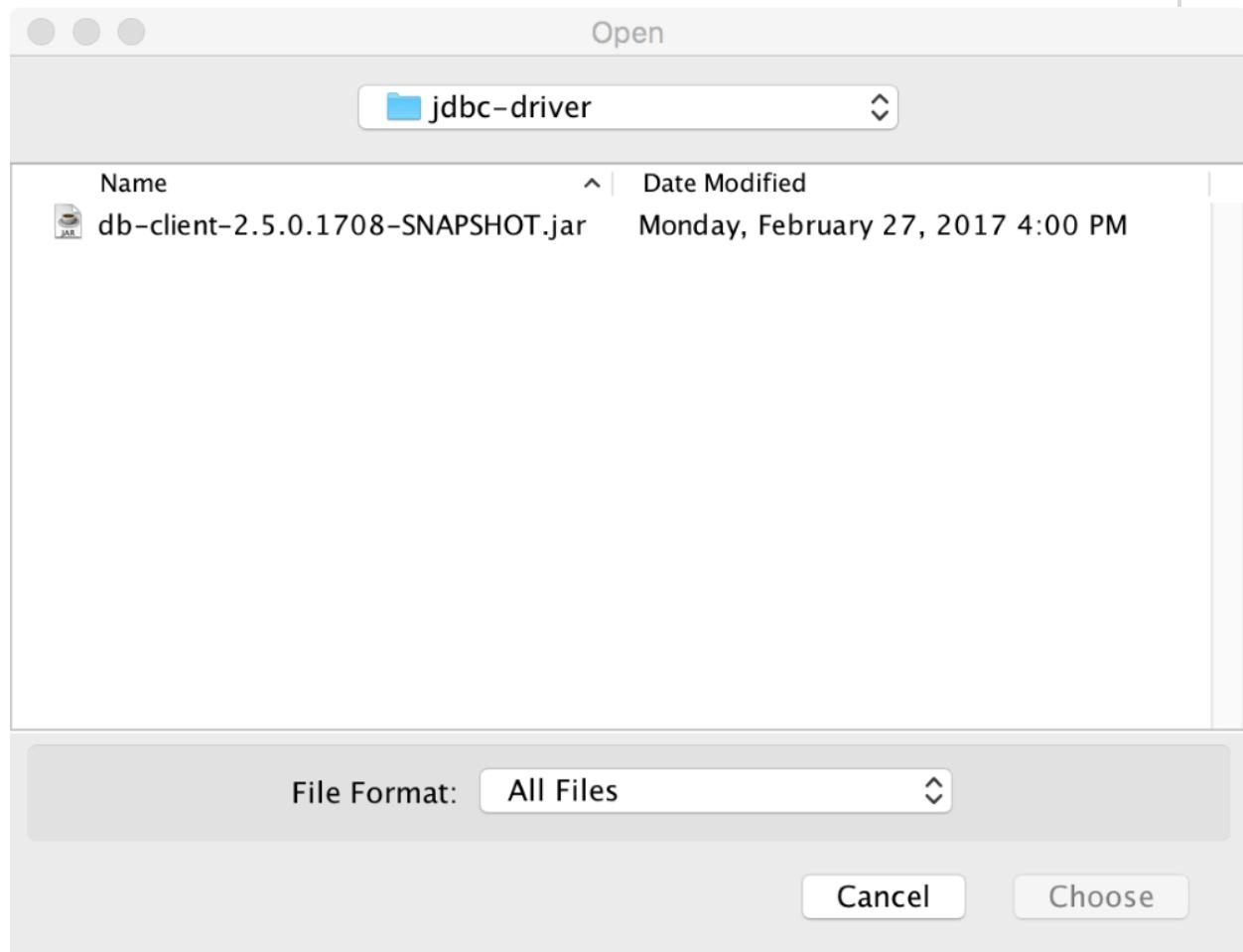
4. In the **Drivers** tab, click the blue + sign **Create a New Driver** icon to display the **Add Driver** window.
 - a. Name the driver and enter `jdbc:splice://localhost:1527/splicedb` in the Example URL field:



Use `localhost:1527` with the standalone (local computer) version of splicemachine. If you're running Splice Machine on a cluster, substitute the address of your server for `localhost`; for example:
`jdbc:splice://mySrv123cba:1527/spicedb`.

- b. Click the Extra Class Path button, and click the Add button.

- c. Navigate to and select the Splice JDBC Driver jar file, which you'll find it in the `jdbc-driver` folder under the `splicemachine` directory on your computer.



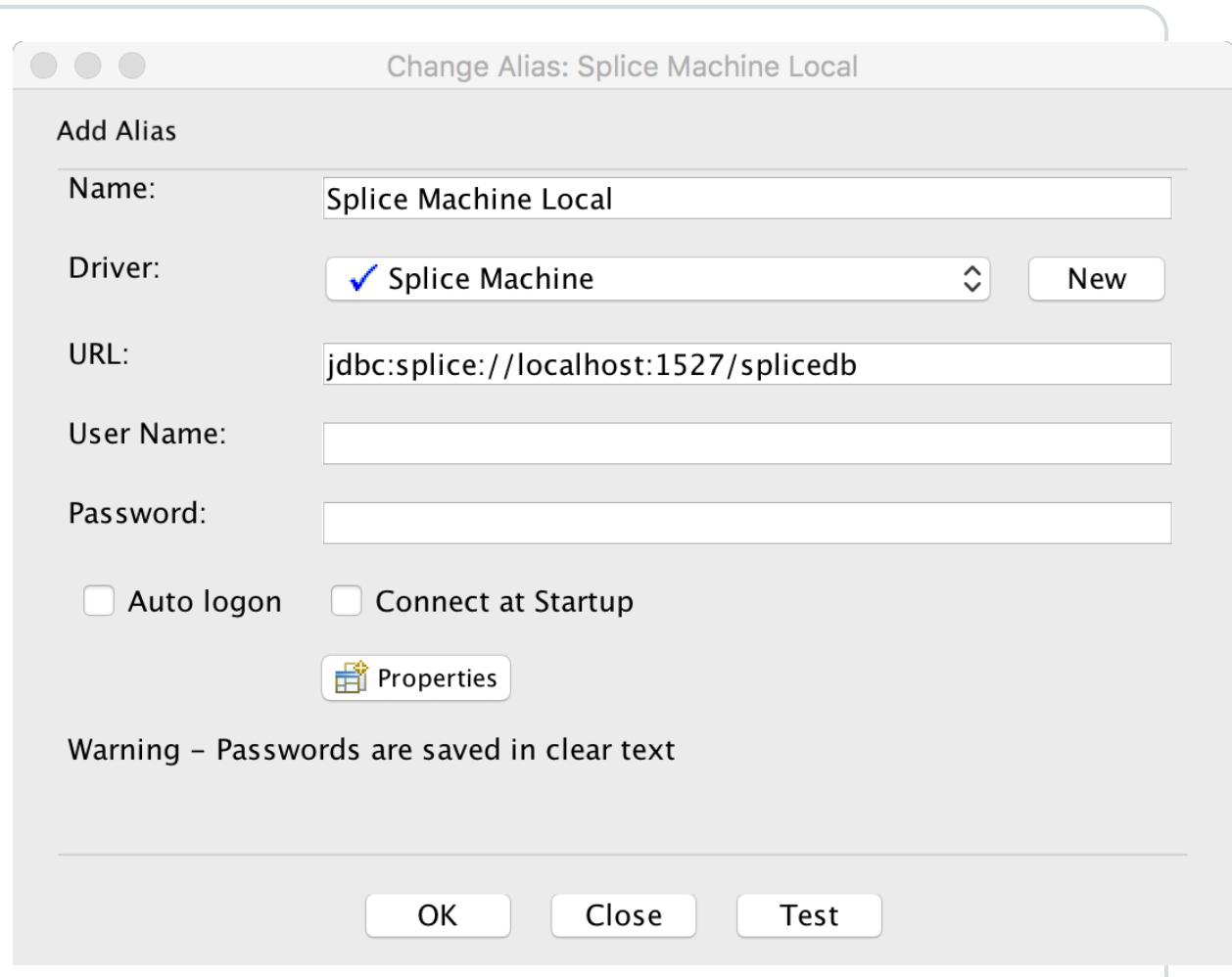
- d. Now, back in the Add Driver screen, click the List Drivers button verify that you see the Splice Machine driver:

```
com.splicemachine.db.jdbc.ClientDriver
```

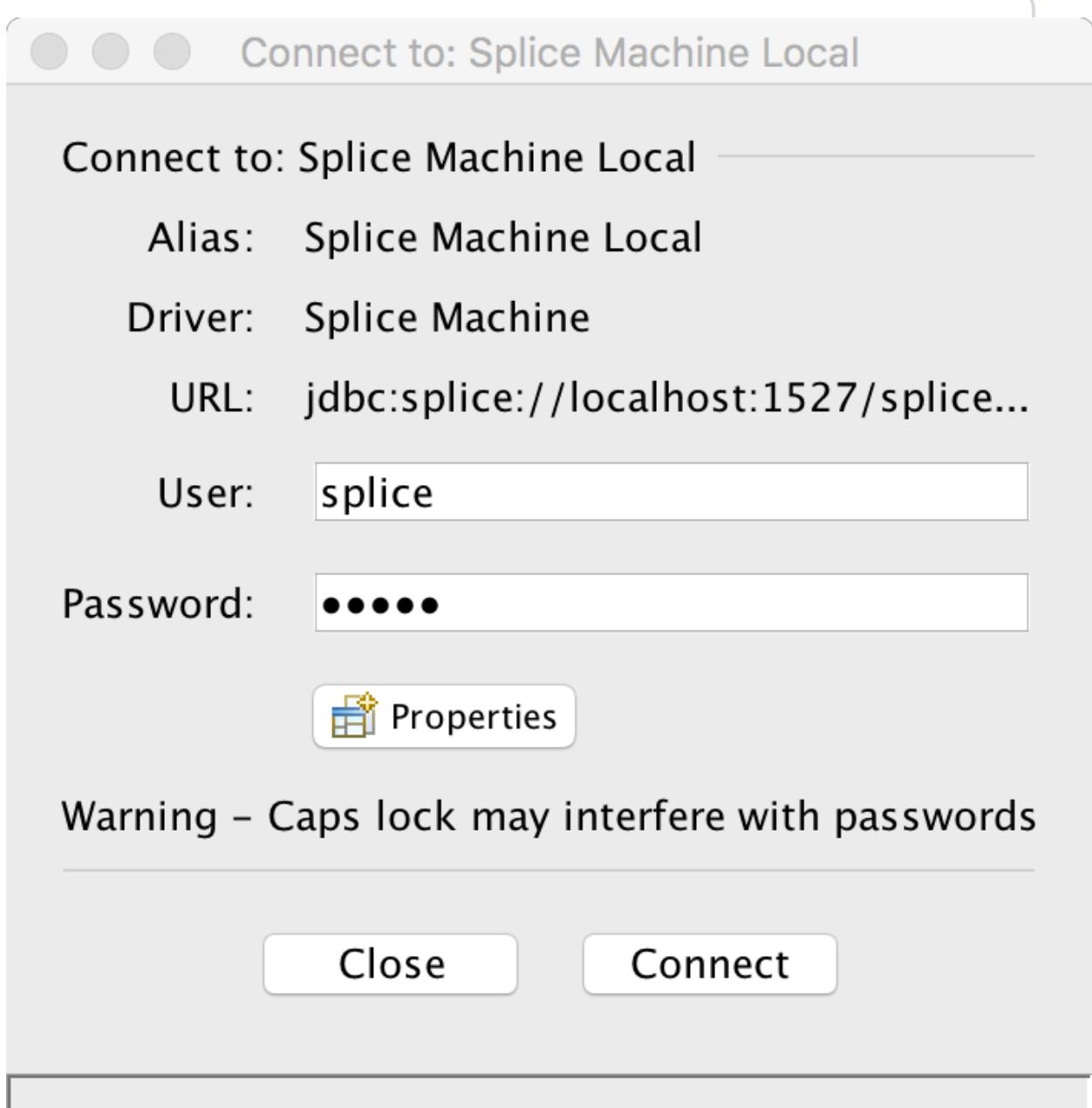
- e. Click the OK button to add the driver entry in SQuirreL.

5. Create a connection alias in SQuirreL

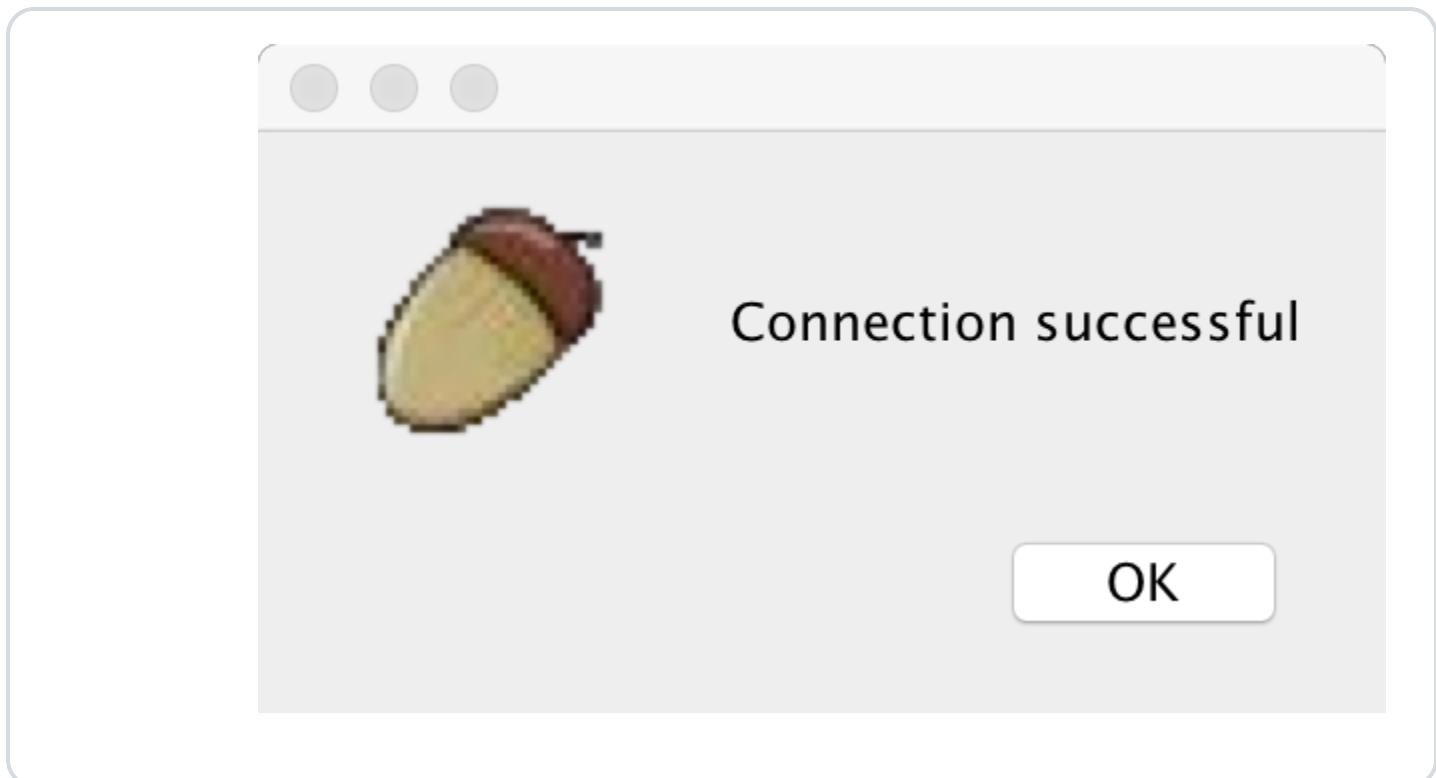
- Click the *Aliases* tab in the SQuirreL window, and then click the *Create new Alias* (blue + sign) button.
- Enter a name for your alias and select the driver you just created from the drop-down list



- c. Click the **Test** button to verify your connection. In the Connect screen, enter `splice` as the `User:` value and `admin` for the `Password:` value.



- d. Click the Connect button to verify your connection. You should see the success message:



Connecting Tableau with Splice Machine Using ODBC

This topic shows you how to connect Tableau to Splice Machine using our ODBC driver. To complete this tutorial, you need to:

- » Have Tableau installed on your Windows or MacOS computer. You can find directions on the Tableau web site (www.tableau.com); you can also download a free trial version of Tableau from there.
- » Have the Splice Machine ODBC driver installed on your computer. Follow the instructions in our Developer's Guide.

Connect Tableau with Splice Machine

This section walks you through configuring Tableau on a Windows PC to connect with Splice Machine using our ODBC driver.

1. Install Tableau, if you've not already done so

[Follow the instructions on the Tableau web site.](#)

2. Install the Splice Machine ODBC driver

Follow our instructions for installing the driver on Unix or Windows. This includes instructions for setting up your data source (DSN), which we'll use with Tableau.

3. Connect from Tableau:

Follow these steps to connect to your data source in Tableau:

a. Open the list of connections:

Click `Connect to Data` on Tableau's opening screen to reveal the list of possible data connections.

b. Select ODBC:

Scroll to the bottom of the `To a server` list, click `More Servers`, then click `Other Databases (ODBC)`.

c. Select your DSN and connect:

Select the DSN you just created (typically named Splice Machine) when installing our ODBC driver) from the drop-down list, and then click the `Connect` button.

d. Select the schema:

Select the schema you want to work with (`splice`), and then select the `Single Table` option.

e. Select the table to view:

Click the search (magnifying glass) icon, and then select the table you want to view from the drop-down list.

For example, we choose the CUSTOMERS table and specify CUSTOMERS (SPLICE) as the connection name for use in Tableau.

4. After you click **OK**, Tableau is ready to work with your data.

Analytics and Machine Learning With Splice Machine

This section provides tutorials to help you to use analytics and machine learning tools with Splice Machine:

- » Connecting with Apache Zeppelin shows you how to use Zeppelin with Splice Machine.

Connecting with Apache Zeppelin

This is an On-Premise-Only topic! [Learn about our products](#)

This tutorial walks you through connecting your on-premise Splice Machine database with Apache Zeppelin, which is a web-based notebook project currently in incubation at Apache. In this tutorial, you'll learn how to use SQL to query your Splice Machine database from Zeppelin.

NOTE:

Zeppelin is already integrated into the Splice Machine Database-as-Service product; please see our *Using Zeppelin* documentation for more information.

See <https://zeppelin.apache.org/> to learn more about Apache Zeppelin.

You can complete this tutorial by [watching a short video](#), or by [following the written directions](#) below.

Watch the Video

The following video shows you how to connect Splice Machine with Apache Zeppelin..

Written Walk Through

This section walks you through using SQL to query a Splice Machine database with Apache Zeppelin..

1. Install Zeppelin:

If you're running on AWS, you can install the Zeppelin sandbox application; if you're using an on-premise database, we recommend following the [instructions in this video](#).

2. Create a new interpreter to run with Splice:

a. Select the Interpreter tab in Zeppelin.

b. Click the Create button (in the upper right of the Zeppelin window) to create a new interpreter. Fill in the property fields as follows:

<i>Name</i>	Whatever name you like; we're using SpliceMachine
<i>Interpreter</i>	Select jdbc from the drop-down list of interpreter types.
<i>default.url</i>	<code>jdbc:splice:/myServer:1527/splicedb</code> (replace <code>myServer</code> with the name of the server that you're using)
<i>default password</i>	admin
<i>default userId</i>	splice
<i>common.max_count</i>	1000
<i>default.driver</i>	<code>com.splicemachine.db.jdbc.ClientDriver</code>
<i>Artifacts</i>	Insert the path to the Splice Machine jar file; for example: <code>/tmp/db-client-2.5.0.1708-SNAPSHOT.jar</code>

- c. Click the **Save** button to save your interpreter definition.

3. Create a note:

Select the **Notebook** tab in Zeppelin, and then click **+ Create new note**.

- a. Specify a name and click the **Create Note** button.
- b. Enable interpreters for the note. In this case, we move the Splice Machine interpreter to the top of the list, then click the Save button to make it the default interpreter:

Interpreter binding

Bind interpreter for this note. Click to Bind/Unbind interpreter. Drag and drop to reorder interpreters. The first interpreter on the list becomes default. To create/remove interpreters, go to [Interpreter](#) menu.



- c. Create a Zeppelin paragraph (a jdbc action) that calls a stored procedure. The procedure we're calling in this tutorial is named MOVIELENS; it is used to analyze data in a table. In this case, we're using this procedure to report statistics on the Age column in our movie watchers database. This Zeppelin paragraph looks like this:

```
%jdbccall MOVIELENS.ContinuousFeatureReport('movielens.user_demographic
s');
```

The %jdbc specifies that we're creating a paragraph that uses a JDBC interpreter; since we've made the SpliceMachine driver our default JDBC connector, it will be used.

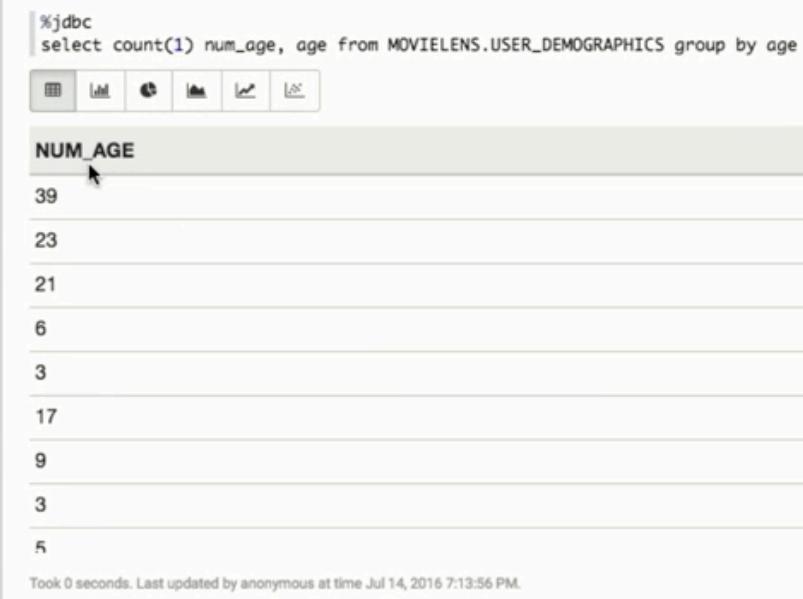
- d. The results of this call look like this:

%jdbc call MOVIELENS.ContinuousFeatureReport('movielens.user_demographics')							FINISHED
COLUMN_NAME	MIN	MAX	COUNT	NUM_NONZEROS	STANDARD_DEVIATION	MEAN	
AGE	7	73	943	943	12	34	

- e. We can also create a new paragraph that performs additional analysis; you'll see that whenever you run a paragraph in Zeppelin, it automatically leaves room at the bottom to create another paragraph.

```
%jdbcselect count(1) num_age, age from MOVIELENS.USER_DEMOGRAPHICS grou
p by age;
```

The results of this paragraph:



The screenshot shows a JDBC cell in Apache Zeppelin. The code is:

```
%jdbc  
select count(1) num_age, age from MOVIELENS.USER_DEMOGRAPHICS group by age
```

The interface includes a toolbar with icons for cell, table, chart, and settings. Below the toolbar is a table with two columns: NUM_AGE and AGE. The data is:

NUM_AGE	AGE
39	30
23	44
21	40
6	52
3	65
17	34
9	57
3	61
5	13

At the bottom, it says "Took 0 seconds. Last updated by anonymous at time Jul 14, 2016 7:13:56 PM."

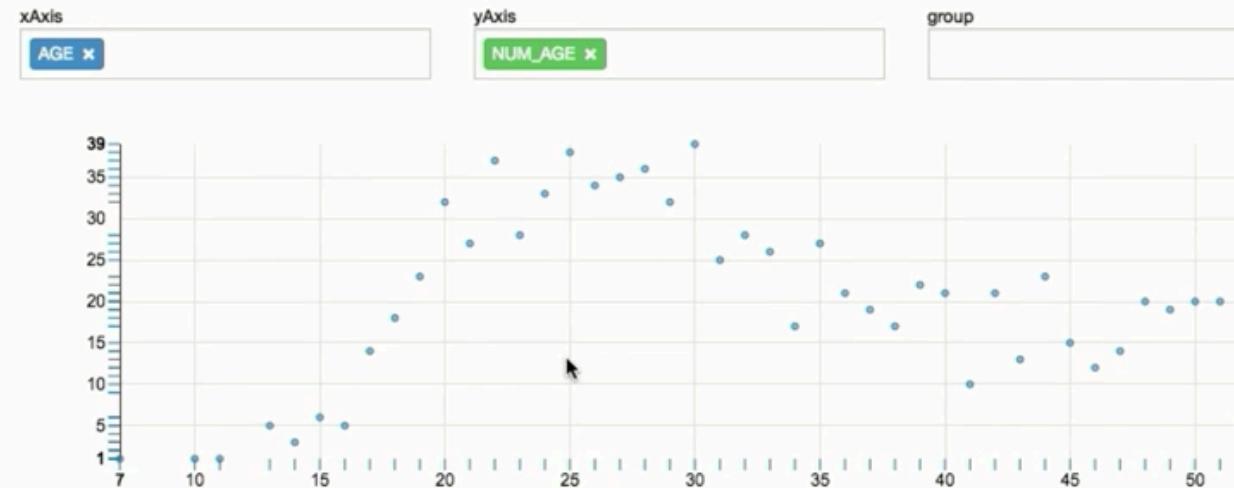
4. Change how you view your data

To get a better sense of what you can do with Zeppelin, we'll modify how we visualize this data:

- Click the rightmost settings icon, then click **settings**.



- Move age to the xAxiS, and the number of people of that age to the yAxiS.
- You'll now see the distribution of ages:



Took 0 seconds. Last updated by anonymous at time Jul 14, 2016 7:14:27 PM. (outdated)

- d. Click the graphs button to select other data visualizations:



SQL Reference Manual

This section contains reference information for Splice Machine SQL. Our implementation includes all of ANSI SQL-99 (SQL3), with added optimizations and features.

Note that this section is modeled on and borrows heavily from the SQL Reference section of the Apache Derby 10.9 documentation, as permitted by the Apache License. This SQL Reference Manual contains the following sections:

Section	Description
Identifiers	Describes the different identifiers used in SQL.
Data Types	Describes the data types used in SQL.
Statements	Reference pages for our implementation of each SQL statement.
Clauses	Reference pages for our implementation of SQL clauses.
Expressions	Describes the expressions you can use in SQL.
Join Operations	Reference pages for our implementation of SQL join operations.
Queries	Reference pages for our implementation of SQL queries.
SQL Built-in Functions	Reference pages for the standard SQL functions featured in our implementation.
Built-in System Procedures and Functions	Reference pages for Splice Machine system procedures and functions.
System Tables	Descriptions of the system tables.
Argument Matching	Describes how Splice Machine matches Java data types and methods with arguments supplied in SQL statements.
SQL Limitations	A summary of various size limitations in Splice Machine.
Reserved Words	A list of the reserved words in Splice Machine.

For a summary of all Splice Machine documentation, see the Documentation Summary topic.

Acknowledgment

Since the Apache Derby documentation served as a starting point for this documentation, **Splice Machine would like to acknowledge the contribution of the Apache Derby community** to the Splice Machine product and documentation.

Splice Machine SQL Summary

This topic summarizes the SQL-99+ features in Splice Machine SQL and some of the [SQL optimizations](#) that our database engine performs.

SQL Feature Summary

This table summarizes some of the ANSI SQL-99+ features available in Splice Machine:

Feature	Examples
<i>Aggregation functions</i>	AVG, COUNT, MAX, MIN, STDDEV_POP, STDDEV_SAMP, SUM
<i>Conditional functions</i>	CASE, searched CASE
<i>Data Types</i>	INTEGER, REAL, CHARACTER, DATE, BOOLEAN, BIGINT
<i>DDL</i>	CREATE TABLE, CREATE SCHEMA, CREATE INDEX, ALTER TABLE, DELETE, UPDATE
<i>DML</i>	INSERT, DELETE, UPDATE, SELECT
<i>Isolation Levels</i>	Snapshot isolation
<i>Joins</i>	INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN
<i>Predicates</i>	IN, BETWEEN, LIKE, EXISTS
<i>Privileges</i>	Privileges for SELECT, DELETE, INSERT, EXECUTE
<i>Query Specification</i>	SELECT DISTINCT, GROUP BY, HAVING
<i>SET functions</i>	UNION, ABS, MOD, ALL, CHECK
<i>String functions</i>	CHAR, Concatenation (), INSTR, LCASE (LOWER), LENGTH, LTRIM, REGEXP_LIKE, REPLACE, RTRIM, SUBSTR, UCASE (UPPER), VARCHAR
<i>Sub-queries</i>	Yes
<i>Transactions</i>	COMMIT, ROLLBACK

Feature	Examples
<i>Triggers</i>	Yes
<i>User-defined functions (UDFs)</i>	Yes
Views	Including grouped views
<i>Window functions</i>	AVG, COUNT, DENSE_RANK, FIRST_VALUE, LAG, LAST_VALUE, LEAD, MAX, MIN, RANK, ROW_NUMBER, STDDEV_POP, STDDEV_SAMP, SUM

SQL Optimizations

Splice Machine performs a number of SQL optimizations that enhance the processing speed of your queries:

- » typed columns
- » sparse columns
- » flexible schema
- » secondary indices
- » real-time asynchronous statistics
- » cost-based optimizer

SQL Limitations

This topic specifies limitations for various values in Splice Machine SQL:

- » [Database Value Limitations](#)
- » [Date, Time, and TimeStamp Limitations](#)
- » [Identifier Length Limitations](#)
- » [Numeric Limitations](#)
- » [String Limitations](#)
- » [XML Limitations](#)

Database Value Limitations

The following table lists limitations on various database values in Splice Machine.

Value	Limit
Maximum columns in a table	100000
Maximum columns in a view	5000
Maximum number of parameters in a stored procedure	90
Maximum indexes on a table	32767 or storage capacity
Maximum tables referenced in an SQL statement or a view	Storage capacity
Maximum elements in a select list	1012
Maximum predicates in a WHERE or HAVING clause	Storage capacity
Maximum number of columns in a GROUP BY clause	32677
Maximum number of columns in an ORDER BY clause	1012
Maximum number of prepared statements	Storage capacity
Maximum declared cursors in a program	Storage capacity
Maximum number of cursors opened at one time	Storage capacity
Maximum number of constraints on a table	Storage capacity

Value	Limit
Maximum level of subquery nesting	Storage capacity
Maximum number of subqueries in a single statement	Storage capacity
Maximum number of rows changed in a unit of work	Storage capacity
Maximum constants in a statement	Storage capacity
Maximum depth of cascaded triggers	16

Date, Time, and TimeStamp Limitations

The following table lists limitations on date, time, and timestamp values in Splice Machine.

Value	Limit
Smallest DATE value	0001-01-01
Largest DATE value	9999-12-31
Smallest TIME value	00:00:00
Largest TIME value	24:00:00
Smallest TIMESTAMP value	1677-09-21-00.12.44.000000
Largest TIMESTAMP value	2262-04-11-23.47.16.999999

Identifier Length Limitations

The following table lists limitations on identifier lengths in Splice Machine.

Identifier	Maximum Number of Characters Allowed
Constraint name	128
Correlation name	128
Cursor name	128

Identifier	Maximum Number of Characters Allowed
Data source column name	128
Data source index name	128
Data source name	128
Savepoint name	128
Schema name	128
Unqualified column name	128
Unqualified function name	128
Unqualified index name	128
Unqualified procedure name	128
Parameter name	128
Unqualified trigger name	128
Unqualified table name, view name, stored procedure name	128

Numeric Limitations

The following lists limitations on the numeric values in Splice Machine.

Value	Limit
Smallest INTEGER	-2,147,483,648
Largest INTEGER	2,147,483,647
Smallest BIGINT	-9,223,372,036,854,775,808
Largest BIGINT	9,223,372,036,854,775,807
Smallest SMALLINT	-32,768
Largest SMALLINT	32,767
Largest decimal precision	31

Value	Limit
Smallest DOUBLE	-1.79769E+308
Largest DOUBLE	1.79769E+308
Smallest positive DOUBLE	2.225E-307
Largest negative DOUBLE	-2.225E-307
Smallest REAL	-3.402E+38
Largest REAL	3.402E+38
Smallest positive REAL	1.175E-37
Largest negative REAL	-1.175E-37

String Limitations

The following table lists limitations on string values in Splice Machine.

Value	Maximum Limit
Length of CHAR	254 characters
Length of VARCHAR	32,672 characters
Length of LONG VARCHAR	32,670 characters
Length of CLOB*	2,147,483,647 characters
Length of BLOB*	2,147,483,647 characters
Length of character constant	32,672
Length of concatenated character string	2,147,483,647
Length of concatenated binary string	2,147,483,647
Number of hex constant digits	16,336
Length of DOUBLE value constant	30 characters

Value	Maximum Limit
	* If you're using our 32-bit ODBC driver, CLOB and BLOB objects are limited to 512 MB in size, instead of 2 GB , due to address space limitations.

Reserved Words

This section lists all of the Splice Machine reserved words, including those in the SQL standard. Splice Machine will return an error if you use any of these keywords as an identifier name unless you surround the identifier name with quotes (""). See SQL Identifier Syntax.

Reserved Word
ADD
ALL
ALLOCATE
ALTER
AND
ANY
ARE
AS
ASC
ASSERTION
AT
AUTHORIZATION
AVG
BEGIN
BETWEEN
BIGINT
BIT
BOOLEAN
BOTH
BY

Reserved Word
CALL
CASCADE
CASCADED
CASE
CAST
CHAR
CHARACTER
CHECK
CLOSE
COALESCE
COLLATE
COLLATION
COLUMN
COMMIT
CONNECT
CONNECTION
CONSTRAINT
CONSTRAINTS
CONTINUE
CONVERT
CORRESPONDING
CREATE
CROSS
CURRENT

Reserved Word
CURRENT_DATE
CURRENT_ROLE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DELETE
DESC
DESCRIBE
DIAGNOSTICS
DISCONNECT
DISTINCT
DOUBLE
DROP
ELSE
END
END-EXEC

Reserved Word
ESCAPE
EXCEPT
EXCEPTION
EXEC
EXECUTE
EXISTS
EXPLAIN
EXTERNAL
FALSE
FETCH
FIRST
FLOAT
FOR
FOREIGN
FOUND
FROM
FULL
FUNCTION
GET
GETCURRENTCONNECTION
GLOBAL
GO
GOTO
GRANT

Reserved Word
GROUP
HAVING
HOUR
IDENTITY
IMMEDIATE
IN
INDICATOR
INITIALLY
INNER
INOUT
INPUT
INSENSITIVE
INSERT
INT
INTEGER
INTERSECT
INTO
IS
ISOLATION
JOIN
KEY
LAST
LEFT
LIKE

Reserved Word
LOWER
LTRIM
MATCH
MAX
MIN
MINUTE
NATIONAL
NATURAL
NCHAR
NVARCHAR
NEXT
NO
NONE
NOT
NULL
NULLIF
NUMERIC
OF
ON
ONLY
OPEN
OPTION
OR
ORDER

Reserved Word
OUTER
OUTPUT
OVER
OVERLAPS
PAD
PARTIAL
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVILEGES
PROCEDURE
PUBLIC
READ
REAL
REFERENCES
REGEXP_LIKE
RELATIVE
RESTRICT
REVOKE
RIGHT
ROLLBACK
ROWS
ROW_NUMBER

Reserved Word
RTRIM
SCHEMA
SCROLL
SECOND
SELECT
SESSION_USER
SET
SMALLINT
SOME
SPACE
SQL
SQLCODE
SQLERROR
SQLSTATE
SUBSTR
SUBSTRING
SUM
TABLE
TEMPORARY
TEXT
TIMEZONE_HOUR
TIMEZONE_MINUTE
TO
TRANSACTION

Reserved Word
TRANSLATE
TRANSLATION
TRIM
TRUE
UNION
UNIQUE
UNKNOWN
UPDATE
UPPER
USER
USING
VALUES
VARCHAR
VARYING
VIEW
WHENEVER
WHERE
WITH
WORK
WRITE
XML
XMLEXISTS
XMLPARSE
XMLQUERY

Reserved Word
XMLSERIALIZE
YEAR

Argument Matching in Splice Machine

When you declare a function or procedure using `CREATE FUNCTION/PROCEDURE`, Splice Machine does not verify whether a matching Java method exists. Instead, Splice Machine looks for a matching method only when you invoke the function or procedure in a later SQL statement.

At that time, Splice Machine searches for a public, static method having the class and method name declared in the `EXTERNAL NAME` clause of the earlier `CREATE FUNCTION/PROCEDURE` statement. Furthermore, the Java types of the method's arguments and return value must match the SQL types declared in the `CREATE FUNCTION/PROCEDURE` statement.

The following may happen:

Result	Description
<i>Success</i>	If exactly one Java method matches, then Splice Machine invokes it.
<i>Ambiguity</i>	If exactly one Java method matches, then Splice Machine invokes it.
<i>Failure</i>	Splice Machine also raises an error if no method matches.

In mapping SQL data types to Java data types, Splice Machine considers the following kinds of matches:

Result	Description	
<i>Primitive Match</i>	Splice Machine looks for a primitive Java type corresponding to the SQL type. For instance, SQL <code>INTEGER</code> matches Java <code>int</code>	
<i>Wrapper Match</i>	Splice Machine looks for a wrapper class in the <code>java.lang</code> or <code>java.sql</code> packages corresponding to the SQL type. For instance, SQL <code>INTEGER</code> matches <code>java.lang.Integer</code> . For a user-defined type (UDT), Splice Machine looks for the UDT's external name class.	
<i>Array Match</i>	For <code>OUT</code> and <code>INOUT</code> procedure arguments, Splice Machine looks for an array of the corresponding primitive or wrapper type. For example, an <code>OUT</code> procedure argument of type SQL <code>INTEGER</code> matches <code>int[]</code> and <code>Integer[]</code> .	
<i>ResultSet Match</i>	If a procedure is declared to return n RESULT SETS, Splice Machine looks for a method whose last n arguments are of type <code>java.sql.ResultSet[]</code> .	

Splice Machine resolves function and procedure invocations as follows:

Call type	Resolution
<i>Function</i>	Splice Machine looks for a method whose argument and return types are <i>primitive matches</i> or <i>wrapper matches</i> for the function's SQL arguments and return value.
<i>Procedure</i>	<p>Splice Machine looks for a method which returns void and whose argument types match as follows:</p> <ul style="list-style-type: none"> » IN - Method arguments are <i>primitive matches</i> or <i>wrapper matches</i> for the procedure's IN arguments. » OUT and INOUT - Method arguments are <i>array matches</i> for the procedure's OUT and INOUT arguments. <p>In addition, if the procedure returns <i>n</i> RESULT SETS, then the last <i>n</i> arguments of the Java method must be of type <code>java.sql.ResultSet[]</code></p>

Example of argument matching

The following function:

```
CREATE FUNCTION TO_DEGREES
    ( RADIANS DOUBLE )
RETURNS DOUBLE
PARAMETER STYLE JAVA
NO SQL LANGUAGE JAVA
EXTERNAL NAME 'example.MathUtils.toDegrees'
;
```

would match all of the following methods:

```
public static double toDegrees( double arg ) {...}
```

Note that Splice Machine raises an exception if it finds more than one matching method.

Mapping SQL data types to Java data types

The following table shows how Splice Machine maps specific SQL data types to Java data types.

SQL and Java type correspondence

SQL Type	Primitive Match	Wrapper Match
BOOLEAN	<code>boolean</code>	<code>java.lang.Boolean</code>
SMALLINT	<code>short</code>	<code>java.lang.Integer</code>
INTEGER	<code>int</code>	<code>java.lang.Integer</code>
BIGINT	<code>long</code>	<code>java.lang.Long</code>
DECIMAL	<code>None</code>	<code>java.math.BigDecimal</code>
NUMERIC	<code>None</code>	<code>java.math.BigDecimal</code>
REAL	<code>float</code>	<code>java.lang.Float</code>
DOUBLE	<code>double</code>	<code>java.lang.Double</code>
FLOAT	<code>double</code>	<code>java.lang.Double</code>
CHAR	<code>None</code>	<code>java.lang.String</code>
VARCHAR	<code>None</code>	<code>java.lang.String</code>
LONG VARCHAR	<code>None</code>	<code>java.lang.String</code>
CLOB	<code>None</code>	<code>java.sql.Clob</code>
BLOB	<code>None</code>	<code>java.sql.Blob</code>
DATE	<code>None</code>	<code>java.sql.Date</code>
TIME	<code>None</code>	<code>java.sql.Time</code>
TIMESTAMP	<code>None</code>	<code>java.sql.Timestamp</code>
User-defined type	<code>None</code>	Underlying Java class

See Also

» [About Data Types](#)

Identifiers

This section contains the reference documentation for the Splice Machine SQL Identifiers, in these topics:

- » This page provides an introduction to `SQLIdentifier`.
- » The SQL Identifier Syntax topic contains additional information about `SQLIdentifier` naming rules, capitalization, and special characters.
- » The SQL Identifier Types topic provides specific information about the different types of `SQLIdentifier`s that you'll find mentioned in this manual, including:
 - AuthorizationIdentifier
 - column-Name and simple-column-Name
 - constraint-Name
 - correlation-Name
 - index-Name
 - new-Table-Name
 - RoleName
 - schemaName
 - synonym-Name
 - table-Name
 - triggerName
 - view-Name

About `SQLIdentifier`

An `SQLIdentifier` is a dictionary object identifier that conforms to the rules of ANSI SQL; identifiers for dictionary objects:

- » are limited to 128 characters
- » are automatically translated into uppercase by the system, making them case-insensitive unless delimited by double quotes
- » cannot be a Splice Machine SQL keyword unless delimited by double quotes
- » can sometimes be qualified by a schema, table, or correlation name, as described below

Examples:

Here is an example of a simple, unqualified `SQLIdentifier` used to name a table:

```
CREATE TABLE Coaches( ID INT NOT NULL );
```

And here's an example of a table name (`Coaches`) qualified by a schema name (`Baseball`):

```
CREATE TABLE Baseball.Coaches( ID INT NOT NULL );
```

This view name is stored in system catalogs as `PITCHINGCOACHES`, since it is not quoted:

```
CREATE VIEW PitchingCoaches(RECEIVED) AS VALUES 1;
```

Whereas this view name is quoted, and thus is stored as `PitchingCoaches` in the system catalog:

```
CREATE VIEW "PitchingCoaches"(RECEIVED) AS VALUES 1;
```



Complete syntax, including information about case sensitivity and special character usage, in SQL Identifier types is found in the SQL Identifier Syntax topic in this section.

Identifier Types

This topic describes the different types of SQLIdentifiers that are used in this manual. .



Complete syntax, including information about case sensitivity and special character usage in SQL Identifier types, is found in the SQL Identifier Syntax topic in this section.

We use a number of different identifier types in the SQL Reference Manual, all of which are `SQLIdentifiers`. Some can be qualified with schema, table, or correlation names, as described in the following table:

Topic	Description
Authorization Identifier	<p>An Authorization Identifier is an <code>SQLIdentifier</code> that represents the name of the user when you specify one in a connection request, otherwise known as a user name. When you connect with a user name, that name becomes the default schema name; if you do not specify a user name in the connect request, the default user name and <code>schemaName</code> is <code>SPLICE</code>.</p> <p>User names can be case-sensitive within the authentication system, but they are always case-insensitive within Splice Machine's authorization system unless they are delimited.</p>
column-Name	<p>A column-Name is a <code>SQLIdentifiers</code> that can be unqualified <code>simple-column-Names</code>. or can be qualified with a <code>table-name</code> or <code>correlation-name</code>.</p> <p>See the Column Name Notes section below for information about when a column-Name can or cannot be qualified.</p>
column-Position	<p>A column-Position is an integer value that specifies the ordinal position value of the column. The first column is column 1.</p>
column-Name-or-Position	<p>A column-Name-or-Position is either a <code>column-Name</code> or <code>column-Position</code> value.</p>
constraint-Name	<p>A constraint-Name is a simple <code>SQLIdentifier</code> used to name constraints.</p> <p>You cannot qualify a constraint-Name.</p>

correlation-Name	<p>A correlation-Name is a simple SQLIdentifier used in a FROM clause as a new name or alias for that table.</p> <p>You cannot qualify a correlation-Name, nor can you use it for a column named in the FOR UPDATE clause, as described in the Correlation Name Notes section below</p>
index-Name	<p>An index-Name is an SQLIdentifier that can be qualified with a schemaName.</p> <p>If you do not use a qualifying schema name, the default schema is assumed. Note that system table indexes are qualified with the SYS. schema prefix.</p>
new-Table-Name	<p>A new-Table-Name is a simple SQLIdentifier that is used when renaming a table with the RENAME TABLE statement.</p> <p>You cannot qualify a new table name with a schema name, because the table already exists in a specific schema.</p>
RoleName	<p>A RoleName is a simple SQLIdentifier used to name roles in your database.</p> <p>You cannot qualify a role name.</p>
schemaName	<p>A schemaName is used when qualifying the names of dictionary objects such as tables and indexes.</p> <p>The default user schema is named SPLICE if you do not specify a user name at connection time, SPLICE is assumed as the schema for any unqualified dictionary objects that you reference.</p> <p>Note that you must always qualify references to system tables with the SYS. prefix, e.g. SYS.SYSROLES.</p>
simple-column-Name	<p>A simple-column-Name is used to represent a column that is not qualified by a tableName or correlation-Name, as described in the Column Name Notes section below.</p>
synonym-Name	<p>A synonym-Name is an SQLIdentifier used for synonyms.</p> <p>You can optionally qualify a synonym-Name with a schemaName. If you do not use a qualifying schema name, the default schema is assumed.</p>

table-Name	A table-Name is an SQLIdentifier use to name tables. You can optionally qualify a table-Name with a schemaName. If you do not use a qualifying schema name, the default schema is assumed. Note that system table names are qualified with the SYS. schema prefix.
triggerName	A triggerName is an SQLIdentifier used to name user-defined triggers. You can optionally qualify a triggerName with a schemaName. If you do not use a qualifying schema name, the default schema is assumed.
view-Name	A view-Name is an SQLIdentifier used to name views. You can optionally qualify a view-Name with a schemaName. If you do not use a qualifying schema name, the default schema is assumed.

Column Name Notes {#Note.ColumnName}

Column names are either simple-column-Name identifiers, which cannot be qualified, or column-Name identifiers that can be qualified with a table-Name or correlation-Name. Here's the syntax:

```
[ { table-Name | correlation-Name } . ] SQLIdentifier
```

In some circumstances, you must use a simple-column-Name and cannot qualify the column name:

- » When creating a table ([CREATE TABLE](#) statement).
- » In a column's correlation-Name in a[SELECT expression](#)
- » In a column'scorrelation-Name in a[TABLE expression](#)

Correlation Name Notes {#Note.CorrelationName}

You cannot use a correlation name for columns that are listed in the FOR UPDATE list of a SELECT. For example, in the following:

```
SELECT Average AS corrCol1, Homeruns AS corrCol2, Strikeouts      FROM Batting      FOR U
PDATE of Average, Strikeouts;
```

- » You cannot use corrCol1 as a correlation name for Average because Average is listed in the FOR UPDATE list.
- » You can use corrCol2 as a correlation name for HomeRuns because the HomeRuns column is not in the update list.

SQL Identifier Syntax

An SQLIdentifier is a dictionary object identifier that conforms to the rules of ANSI SQL; identifiers for dictionary objects:

- » are limited to 128 characters
- » are automatically translated into uppercase by the system, making them case-insensitive unless delimited by double quotes
- » cannot be a Splice Machine SQL keyword unless delimited by double quotes
- » can sometimes be qualified by a schema, table, or correlation name, as described below

Examples:

This view name:

```
CREATE VIEW PitchingCoaches(RECEIVED) AS VALUES 1;
```

is stored in system catalogs as PITCHINGCOACHES, since it is not quoted.

Whereas this view name:

```
CREATE VIEW "PitchingCoaches"(RECEIVED) AS VALUES 1;
```

is quoted, and thus is stored as PitchingCoaches in the system catalog

Qualifying dictionary objects

Since some dictionary objects can be contained within other objects, you can qualify those dictionary object names. Each component is separated from the next by a period (.). You qualify a dictionary object name in order to avoid ambiguity.

Examples:

Here is an example of a simple, unqualified SQLIdentifier used to name a table:

```
CREATE TABLE Coaches( ID INT NOT NULL );
```

And here's an example of a table name (Coaches) qualified by a schema name (Baseball):

```
CREATE TABLE Baseball.Coaches( ID INT NOT NULL );
```

Rules for SQL Identifiers

Here are some additional rules that apply to SQLIdentifiers:

» Ordinary identifiers are identifiers not surrounded by double quotation marks:

- An ordinary identifier must begin with a letter and contain only letters, underscore characters (_), and digits.
 - All Unicode letters and digits are permitted; however, Splice Machine does not attempt to ensure that the characters in identifiers are valid in the database's locale.
- » Delimited identifiers are identifiers surrounded by double quotation marks:
- A delimited identifier can contain any characters within the double quotation marks.
 - The enclosing double quotation marks are not part of the identifier; they serve only to mark its beginning and end.
 - Spaces at the end of a delimited identifier are truncated.
 - You can use two consecutive double quotation marks within a delimited identifier to include a double quotation mark within the identifier.

Capitalization and Special Characters in SQL Statements

You can submit SQL statements to Splice Machine as strings by using JDBC; these strings use the Unicode character set. Within these strings:

- » Double quotation marks delimit special identifiers referred to in ANSI SQL as *delimited identifiers*.
- » Single quotation marks delimit character strings.
- » Within a character string, to represent a single quotation mark or apostrophe, use two single quotation marks. (In other words, a single quotation mark is the escape character for a single quotation mark).
- » SQL keywords are case-insensitive. For example, you can type the keyword `SELECT` as `SELECT`, `Select`, `select`, or `SELECT`.
- » ANSI SQL -style identifiers are case-insensitive unless they are delimited.
- » Java-style identifiers are always case-sensitive.

Other Special Characters:

- » * is a wildcard within a *Select Expression*. It can also be the multiplication operator. In all other cases, it is a syntactical metasymbol that flags items you can repeat 0 or more times.
- » % and _ are character wildcards when used within character strings following a `LIKE` operator (except when escaped with an escape character). See Boolean expressions.
- » Comments can be either single-line or multi-line, as per the ANSI SQL standard:
 - » Single line comments start with two dashes (--) and end with the newline character.
 - » Multi-line comments are bracketed and start with forward slash star (* /), and end with star forward slash (* /). Note that bracketed comments may be nested. Any text between the starting and ending comment character sequence is ignored.

Data Types

The SQL type system is used by the language compiler to determine the compile-time type of an expression and by the language execution system to determine the runtime type of an expression, which can be a subtype or implementation of the compile-time type.

Each type has associated with it values of that type. In addition, values in the database or resulting from expressions can be `NULL`, which means the value is missing or unknown. Although there are some places where the keyword `NULL` can be explicitly used, it is not in itself a value, because it needs to have a type associated with it.

This section contains the reference documentation for the Splice Machine SQL Data Types, in the following subsections:

- » [Character String Data Types](#)
- » [Date and Time Data Types](#)
- » [Large Object Binary \(LOB\) Data Types](#)
- » [Numeric Data Types](#)
- » [Other Data Types](#)

Character String Data Types

These are the character string data types:

Data Type	Description
CHAR	The CHAR data type provides for fixed-length storage of strings.
LONG VARCHAR	The LONG VARCHAR type allows storage of character strings with a maximum length of 32,700 characters. It is identical to VARCHAR, except that you cannot specify a maximum length when creating columns of this type.
VARCHAR	The VARCHAR data type provides for variable-length storage of strings.

Date and Time Data Types

These are the date and time data types:

Data Type	Description
-----------	-------------

DATE	The DATE data type provides for storage of a year-month-day in the range supported by <code>java.sql.Date</code> .
TIME	The TIME data type provides for storage of a time-of-day value.
TIMESTAMP	The TIMESTAMP data type stores a combined DATE and TIME value, and allows a fractional-seconds value of up to nine digits.

Large Object Binary (LOB) Data Types

These are the LOB data types:

Data Type	Description
BLOB	The BLOB (binary large object) data type is used for varying-length binary strings that can be up to 2,147,483,647 characters long.
CLOB	The CLOB (character large object) data type is used for varying-length character strings that can be up to 2,147,483,647 characters long.
TEXT	Exactly the same as CLOB.

Numeric Data Types

These are the numeric data types:

Data Type	Description
BIGINT	The BIGINT data type provides 8 bytes of storage for integer values.
DECIMAL	The DECIMAL data type provides an exact numeric in which the precision and scale can be arbitrarily sized. You can use DECIMAL and NUMERIC interchangeably.

DOUBLE	The DOUBLE data type provides 8-byte storage for numbers using IEEE floating-point notation. DOUBLE PRECISION can be used synonymously with DOUBLE.
DOUBLE PRECISION	The DOUBLE PRECISION data type provides 8-byte storage for numbers using IEEE floating-point notation. DOUBLE can be used synonymously with DOUBLE PRECISION.
FLOAT	The FLOAT data type is an alias for either a REAL or DOUBLE PRECISION data type, depending on the precision you specify.
INTEGER	INTEGER provides 4 bytes of storage for integer values.
NUMERIC	The NUMERIC data type provides an exact numeric in which the precision and scale can be arbitrarily sized. You can use NUMERIC and DECIMAL interchangeably.
REAL	The REAL data type provides 4 bytes of storage for numbers using IEEE floating-point notation.
SMALLINT	The SMALLINT data type provides 2 bytes of storage.

Other Data Types

These are the other data types:

Data Type	Description
BOOLEAN	Provides 1 byte of storage for logical values.

See Also

- » Argument Matching
- » Assignments

- » [BIGINT](#) data type
- » [BLOB](#) data type
- » [BOOLEAN](#) data type
- » [CHAR](#) data type
- » [CLOB](#) data type
- » [DATE](#) data type
- » [DECIMAL](#) data type
- » [DOUBLE](#) data type
- » [DOUBLE PRECISION](#) data type
- » [FLOAT](#) data type
- » [INTEGER](#) data type
- » [LONG VARCHAR](#) data type
- » [NUMERIC](#) data type
- » [REAL](#) data type
- » [SMALLINT](#) data type
- » [TEXT](#) data type
- » [TIME](#) data type
- » [TIMESTAMP](#) data type
- » [VARCHAR](#) data type

BIGINT

The BIGINT data type provides 8 bytes of storage for integer values.

Syntax

BIGINT

Corresponding Compile-time Java Type

java.lang.Long

JDBC Metadata Type (java.sql.Types)

BIGINT

Notes

Here are several usage notes for the BIGINT data type:

- » The minimum value is -9223372036854775808 (java.lang.Long.MIN_VALUE)
- » The maximum value is 9223372036854775807 (java.lang.Long.MAX_VALUE)
- » When mixed with other data types in expressions, the resulting data type follows the rules shown in Numeric type promotion in expressions.
- » An attempt to put an integer value of a larger storage size into a location of a smaller size fails if the value cannot be stored in the smaller-size location. Integer types can always successfully be placed in approximate numeric values, although with the possible loss of some precision. BIGINTs can be stored in DECIMALs if the DECIMAL precision is large enough for the value.

Example

9223372036854775807

BLOB

A BLOB (binary large object) value is a varying-length binary string that can be up to 2GB (2,147,483,647) characters long.

If you're using a BLOB with the 32-bit version of our ODBC driver, the size of the BLOB is limited to 512 MB, due to address space limitations.

Like other binary types, BLOBstrings are not associated with a code page. In addition, BLOBstrings do not hold character data.

Syntax

```
{BLOB | BINARY LARGE OBJECT} [ ( length [{K |M |G}] ) ]
```

length

An unsigned integer constant that specifies the number of characters in the BLOB unless you specify one of the suffixes you see below, which change the meaning of the *length* value. If you do not specify a length value, it defaults to two gigabytes (2,147,483,647).

K

If specified, indicates that the length value is in multiples of 1024 (kilobytes).

M

If specified, indicates that the length value is in multiples of 1024*1024 (megabytes).

G

If specified, indicates that the length value is in multiples of 1024*1024*1024 (gigabytes).

Corresponding Compile-time Java Type

```
java.sql.Blob
```

JDBC Metadata Type (java.sql.Types)

```
BLOB
```

Usage Notes

Use the `getBlob` method on the `java.sql.ResultSet` to retrieve a BLOB handle to the underlying data.

There are a number of restrictions on using BLOB and CLOB / TEXT objects, which we refer to as LOB-types:

- » LOB-types cannot be compared for equality (=) and non-equality (!=, <>).

- » LOB-typed values cannot be ordered, so <, <=, >, >= tests are not supported.
- » LOB-types cannot be used in indexes or as primary key columns.
- » DISTINCT, GROUP BY, and ORDER BY clauses are also prohibited on LOB-types.
- » LOB-types cannot be involved in implicit casting as other base-types.

Example

Using an `INSERT` statement to put BLOB data into a table has some limitations if you need to cast a long string constant to a BLOB. You may be better off using a binary stream, as in the following code fragment.

```

package com.splicemachine.tutorials.blob;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class ExampleInsertBlob {

    /**
     * Example of inserting a blob using JDBC
     *
     * @param args[0] - Image file to insert
     * @param args[1] - JDBC URL - optional - defaults to localhost
     */
    public static void main(String[] args) {
        Connection conn = null;
        Statement statement = null;
        ResultSet rs = null;

        if(args.length == 0) {
            System.out.println("You must pass in an file (like an image) to be loaded");
        }

        try {
            String imageFileToLoad = args[0];

            //Default JDBC Connection String - connects to local database
            String dbUrl = "jdbc:splice://localhost:1527/splicedb;user=splice;password=true";

            //Checks to see if a JDBC URL is passed in
            if(args.length > 1) {
                dbUrl = args[1];
            }

            //For the JDBC Driver - Use the Splice Machine Client Driver
            Class.forName("com.splicemachine.db.jdbc.ClientDriver");

            //Connect to the databae
            conn = DriverManager.getConnection(dbUrl);

            //Create a statement
            statement = conn.createStatement();
        }
    }
}

```

```
//Create a table
statement.execute("CREATE TABLE IMAGES(a INT, test BLOB)");

//Create an input stream
InputStream fin = new FileInputStream(imageFileToLoad);
PreparedStatement ps = conn.prepareStatement("INSERT INTO IMAGES VALUES
(?, ?)");
ps.setInt(1, 1477);

// - set value of input parameter to the input stream
ps.setBinaryStream(2, fin);
ps.execute();

ps.close();

//Lets get the count of records
rs = statement.executeQuery("select count(1) from IMAGES");
if(rs.next()) {
    System.out.println("count=[ " + rs.getInt(1) + " ]");
}

} catch (ClassNotFoundException cne) {
    cne.printStackTrace();
} catch (SQLException se) {
    se.printStackTrace();
} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} finally {
    if(rs != null) {
        try { rs.close(); } catch (Exception ignore) { }
    }
    if(statement != null) {
        try { statement.close(); } catch (Exception ignore) { }
    }
    if(conn != null) {
        try { conn.close(); } catch (Exception ignore) { }
    }
}
```

BOOLEAN

The BOOLEAN data type provides 1 byte of storage for logical values.

Syntax

```
BOOLEAN
```

Corresponding Compile-time Java Type

```
java.lang.Boolean
```

JDBC Metadata Type (java.sql.Types)

```
BOOLEAN
```

Usage Notes

Here are several usage notes for the BOOLEAN data type:

- » The legal values are `true`, `false`, and `null`.
- » BOOLEAN values can be cast to and from character type values.
- » For comparisons and ordering operations, `true` sorts higher than `false`.

Examples

```
values true;
values false;
values cast (null as boolean);
```

CHAR

The CHAR data type provides for fixed-length storage of strings.

Syntax

```
CHAR[ACTER] [(length)]
```

length

An unsigned integer literal designating the length in bytes. The default *length* for a CHAR is 1; the maximum size of *length* is 254..

Corresponding Compile-time Java Type

```
java.lang.String
```

JDBC Metadata Type (java.sql.Types)

```
CHAR
```

Usage Notes

Here are several usage notes for the CHAR data type:

- » Splice Machine inserts spaces to pad a string value shorter than the expected length, and truncates spaces from a string value longer than the expected length. Characters other than spaces cause an exception to be raised. When comparison boolean operators are applied to CHARs, the shorter string is padded with spaces to the length of the longer string.
- » When CHARs and VARCHARs are mixed in expressions, the shorter value is padded with spaces to the length of the longer value.
- » The type of a string constant is CHAR.

Examples

```
-- within a string constant use two single quotation marks
-- to represent a single quotation mark or apostrophe
VALUES 'hello this is Joe''s string';

-- create a table with a CHAR field
CREATE TABLE STATUS (
    STATUSCODE CHAR(2) NOT NULL
        CONSTRAINT PK_STATUS PRIMARY KEY,
    STATUSDESC VARCHAR(40) NOT NULL
);
```

CLOB

A CLOB (character large object) value can be up to 2 GB (2,147,483,647 characters) long. A CLOB is used to store unicode character-based data, such as large documents in any character set.

If you're using a CLOB with the 32-bit version of our ODBC driver, the size of the CLOB is limited to 512 MB, due to address space limitations.

Note that, in Splice Machine, TEXT is a synonym for CLOB, and that the documentation for the [TEXT](#) data type functionally matches the documentation for this topic. Splice Machine simply translates TEXT into CLOB.

Syntax

```
{CLOB | CHARACTER LARGE OBJECT} [ ( length [{K |M |G}] ) ]
```

length

An unsigned integer constant that specifies the number of characters in the CLOB unless you specify one of the suffixes you see below, which change the meaning of the *length* value. If you do not specify a length value, it defaults to two giga-characters (2,147,483,647).

K

If specified, indicates that the length value is in multiples of 1024 (kilo-characters).

M

If specified, indicates that the length value is in multiples of 1024*1024 (mega-characters).

G

If specified, indicates that the length value is in multiples of 1024*1024*1024 (giga-characters).

Corresponding Compile-time Java Type

```
java.sql.Clob
```

JDBC Metadata Type ([java.sql.Types](#))

```
CLOB
```

Usage Notes

Use the `getBlob` method on the `java.sql.ResultSet` to retrieve a CLOB handle to the underlying data.

There are a number of restrictions on using using BLOB and CLOB / TEXT objects, which we refer to as LOB-types:

- » LOB-types cannot be compared for equality (=) and non-equality (!=, <>).
- » LOB-typed values cannot be ordered, so <, <=, >, >= tests are not supported.
- » LOB-types cannot be used in indexes or as primary key columns.
- » DISTINCT, GROUP BY, and ORDER BY clauses are also prohibited on LOB-types.
- » LOB-types cannot be involved in implicit casting as other base-types.

Example

```
CREATE TABLE myTable( largeCol CLOB(65535));
```

DATE

The DATE data type provides for storage of a year-month-day in the range supported by *java.sql.Date*.

Syntax

DATE

Corresponding Compile-time Java Type

java.sql.Date

JDBC Metadata Type (*java.sql.Types*)

DATE

Usage Notes

Here are several notes about using the DATE data type:

- » Dates, timestamps must not be mixed with one another in expressions.
- » Any value that is recognized by the *java.sql.Date* method is permitted in a column of the corresponding SQL date/time data type. Splice Machine supports the following formats for DATE:

yyyy-mm-dd
mm/dd/yyyy
dd.mm.yyyy

- » The first of the three formats above is the *java.sql.Date* format.
- » The year must always be expressed with four digits, while months and days may have either one or two digits.
- » Splice Machine also accepts strings in the locale specific date-time format, using the locale of the database server. If there is an ambiguity, the built-in formats above take precedence.

Please see [Working With Date and Time Values](#) in the for information about using simple arithmetic with DATE values.

Examples

```
VALUES DATE('1994-02-23');  
VALUES '1993-09-01';
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE function](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)
- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [Working with Dates](#) in the *Developer's Guide*

DECIMAL

The DECIMAL data type provides an exact numeric in which the precision and scale can be arbitrarily sized. You can specify the *precision* (the total number of digits, both to the left and the right of the decimal point) and the *scale* (the number of digits of the fractional component). The amount of storage required depends on the precision you specify.

Note that NUMERIC is a synonym for DECIMAL, and that the documentation for the [NUMERIC](#) data type exactly matches the documentation for this topic.

Syntax

```
{ DECIMAL | DEC } [(precision [, scale])]
```

precision

Must be between 1 and 31. If not specified, the default precision is 5.

scale

Must be less than or equal to the precision. If not specified, the default scale is 0.

Usage Notes

Here are several notes about using the DECIMAL data type:

- » An attempt to put a numeric value into a DECIMAL is allowed as long as any non-fractional precision is not lost. When truncating trailing digits from a DECIMAL value, Splice Machine rounds down. For example:

```
-- this cast loses only fractional precision
values cast (1.798765 AS decimal(5,2));
1
-----
1.79
      -- this cast does not fit:
values cast (1798765 AS decimal(5,2));
ERROR 22003: The resulting value is outside the range for the data type DECIMAL/NUMERIC(5,2).
```

- » When mixed with other data types in expressions, the resulting data type follows the rules shown in Storing values of one numeric data type in columns of another numeric data type.
- » When two decimal values are mixed in an expression, the scale and precision of the resulting value follow the rules shown in Scale for decimal arithmetic.
- » Integer constants too big for BIGINT are made DECIMAL constants.

Corresponding Compile-time Java Type

java.math.BigDecimal

JDBC Metadata Type (java.sql.Types)

DECIMAL

Examples

```
VALUES 123.456;  
VALUES 0.001;
```

DOUBLE

The DOUBLE data type provides 8-byte storage for numbers using IEEE floating-point notation. DOUBLE PRECISION can be used synonymously with DOUBLE, and the documentation for this topic is identical to the documentation for the [DOUBLE PRECISION](#) topic.

Syntax

DOUBLE

or, alternately

DOUBLE PRECISION

Usage Notes

Here are several usage notes for the DOUBLE/DOUBLE PRECISION data type:

- » The following range limitations apply:

Limit type	Limitation
Smallest DOUBLE value	-1.79769E+308
Largest DOUBLE value	1.79769E+308
Smallest positive DOUBLE value	2.225E-307
Largest negative DOUBLE value	-2.225E-307

NOTE: These limits are different from the `java.lang.Double` Java type limits.

- » An exception is thrown when any double value is calculated or entered that is outside of these value ranges. Arithmetic operations **do not** round their resulting values to zero. If the values are too small, you will receive an exception.
- » Numeric floating point constants are limited to a length of 30 characters.

```
-- this example will fail because the constant is too long:  
values 01234567890123456789012345678901e0;
```

- » When mixed with other data types in expressions, the resulting data type follows the rules shown in [Storing values of one numeric data type in columns of another numeric data type](#).

Corresponding Compile-time Java Type

java.lang.Double

JDBC Metadata Type (java.sql.Types)

DOUBLE

Examples

```
3421E+09  
425.43E9  
9E-10  
4356267544.32333E+30
```

DOUBLE PRECISION

The DOUBLE PRECISION data type provides 8-byte storage for numbers using IEEE floating-point notation. DOUBLE can be used synonymously with DOUBLE PRECISION, and the documentation for this topic is identical to the documentation for the [DOUBLE](#) topic.

Syntax

DOUBLE PRECISION

or, alternately

DOUBLE

Usage Notes

Here are several usage notes for the DOUBLE/DOUBLE PRECISION data type:

- » The following range limitations apply:

Limit type	Limitation
Smallest DOUBLE value	-1.79769E+308
Largest DOUBLE value	1.79769E+308
Smallest positive DOUBLE value	2.225E-307
Largest negative DOUBLE value	-2.225E-307

NOTE: These limits are different from the `java.lang.Double` Java type limits

- » An exception is thrown when any double value is calculated or entered that is outside of these value ranges. Arithmetic operations **do not** round their resulting values to zero. If the values are too small, you will receive an exception.
- » Numeric floating point constants are limited to a length of 30 characters.

```
-- this example will fail because the constant is too long:  
values 01234567890123456789012345678901e0;
```

- » When mixed with other data types in expressions, the resulting data type follows the rules shown in [Storing values of one numeric data type in columns of another numeric data type](#).

Corresponding Compile-time Java Type

java.lang.Double

JDBC Metadata Type (java.sql.Types)

DOUBLE

Examples

```
3421E+09  
425.43E9  
9E-10  
4356267544.32333E+30
```

FLOAT

The FLOAT data type is an alias for either a `DOUBLE PRECISION` data type, depending on the precision you specify.

Syntax

```
FLOAT [ (precision) ]
```

precision

The default precision for FLOAT is 52, which is equivalent to `DOUBLE PRECISION`.

A precision of 23 or less makes FLOAT equivalent to `REAL`.

A precision of 24 or greater makes FLOAT equivalent to `DOUBLE PRECISION`.

If you specify a precision of 0, you get an error. If you specify a negative precision, you get a syntax error.

JDBC Metadata Type (`java.sql.Types`)

```
REAL or DOUBLE
```

Usage Notes

If you are using a precision of 24 or greater, the limits of FLOAT are similar to the limits of `DOUBLE`.

If you are using a precision of 23 or less, the limits of FLOAT are similar to the limits of `REAL`.

Data defined with type `double` at this time. Note that this does not cause a loss of precision, though the data may require slightly more space.

INTEGER Data Type

The INTEGER data type provides 4 bytes of storage for integer values.

Syntax

```
{ INTEGER | INT }
```

Corresponding Compile-Time Java Type

```
java.lang.Integer
```

JDBC Metadata Type (java.sql.Types)

```
INTEGER
```

Minimum Value

```
-2147483648 (java.lang.Integer.MIN_VALUE)
```

Maximum Value

```
2147483647 (java.lang.Integer.MAX_VALUE)
```

Usage Notes

When mixed with other data types in expressions, the resulting data type follows the rules shown in Numeric type promotion in expressions.

See also Storing values of one numeric data type in columns of another numeric data type.

Examples

```
3453  
425
```


LONG VARCHAR

The LONG VARCHAR type allows storage of character strings with a maximum length of 32,670 characters. It is almost identical to VARCHAR, except that you cannot specify a maximum length when creating columns of this type.

Syntax

```
LONG VARCHAR
```

Corresponding Compile-time Java Type

```
java.lang.String
```

JDBC Metadata Type (java.sql.Types)

```
LONGVARCHAR
```

Usage Notes

When you are converting from Java values to SQL values, no Java type corresponds to LONG VARCHAR.

NUMERIC Data Type

NUMERIC is a synonym for the [DECIMAL](#) data type and behaves the same way. The documentation below is a mirror of the documentation for the DECIMAL data type.

NUMERIC provides an exact numeric in which the precision and scale can be arbitrarily sized. You can specify the *precision* (the total number of digits, both to the left and the right of the decimal point) and the *scale* (the number of digits of the fractional component). The amount of storage required depends on the precision you specify.

Syntax

```
NUMERIC [(precision [, scale ])]
```

precision

Must be between 1 and 31. If not specified, the default precision is 5.

scale

Must be less than or equal to the precision. If not specified, the default scale is 0.

Usage Notes

Here are several notes about using the NUMERIC data type:

- » An attempt to put a numeric value into a NUMERIC is allowed as long as any non-fractional precision is not lost. When truncating trailing digits from a NUMERIC value, Splice Machine rounds down. For example:

```
-- this cast loses only fractional precision
values cast (1.798765 AS numeric(5,2));
1
-----
1.79
      -- this cast does not fit:
values cast (1798765 AS numeric(5,2));
ERROR 22003: The resulting value is outside the range for the data type DECIMA
L/NUMERIC(5,2).
```

- » When mixed with other data types in expressions, the resulting data type follows the rules shown in Storing values of one numeric data type in columns of another numeric data type.
- » When two numeric values are mixed in an expression, the scale and precision of the resulting value follow the rules shown in Scale for decimal arithmetic.
- » Integer constants too big for BIGINT are made NUMERIC constants.

Corresponding Compile-time Java Type

`java.math.BigDecimal`

JDBC Metadata Type (`java.sql.Types`)

NUMERIC

Examples

```
VALUES 123.456;  
VALUES 0.001;
```

REAL

The REAL data type provides 4 bytes of storage for numbers using IEEE floating-point notation.

Syntax

REAL

Corresponding Compile-time Java Type

java.lang.Float

JDBC Metadata Type (java.sql.Types)

REAL

Limitations

REAL value ranges:

Limit type	Limit value
<i>Smallest REAL value</i>	-3.402E+38
<i>Largest REAL value</i>	3.402E+38
<i>Smallest positive REAL value</i>	1.175E-37
<i>Largest negative REAL value</i>	-1.175E-37

NOTE: These limits are different from the java.lang.Float Java type limits.

Usage Notes

Here are several usage notes for the REAL data type:

- » An exception is thrown when any double value is calculated or entered that is outside of these value ranges. Arithmetic operations **do not** round their resulting values to zero. If the values are too small, you will receive an exception. The arithmetic operations take place with double arithmetic in order to detect under flows.
- » Numeric floating point constants are limited to a length of 30 characters.

```
-- this example will fail because the constant is too long:  
values 01234567890123456789012345678901e0;
```

- » When mixed with other data types in expressions, the resulting data type follows the rules shown in Numeric type promotion in expressions.
- » See also Storing values of one numeric data type in columns of another numeric data type.
- » Constants always map to [DOUBLE PRECISION](#); use a CAST to convert a constant to a REAL.

SMALLINT

The SMALLINT data type provides 2 bytes of storage.

Syntax

SMALLINT

Corresponding Compile-time Java Type

java.lang.Short

JDBC Metadata Type (java.sql.Types)

SMALLINT

Usage Notes

Here are several usage notes for the SMALLINT data type:

- » The minimum value is `-32768 (java.lang.Short.MIN_VALUE)`.
- » The maximum value is `32767 (java.lang.Short.MAX_VALUE)`.
- » When mixed with other data types in expressions, the resulting data type follows the rules shown in Numeric type promotion in expressions.
- » See also Storing values of one numeric data type in columns of another numeric data type.
- » Constants in the appropriate format always map to INTEGER or BIGINT, depending on their length.

TEXT

A TEXT (character large object) value can be up to 2,147,483,647 characters long. A TEXT object is used to store unicode character-based data, such as large documents in any character set.

Note that, in Splice Machine, TEXT is a synonym for CLOB, and that the documentation for the [CLOB](#) data type functionally matches the documentation for this topic. Splice Machine simply translates TEXT into CLOB.

Syntax

```
TEXT [ ( length [{K |M |G}] ) ]
```

length

An unsigned integer constant that specifies the number of characters in the TEXT unless you specify one of the suffixes you see below, which change the meaning of the *length* value. If you do not specify a length value, it defaults to two giga-characters (2,147,483,647).

K

If specified, indicates that the length value is in multiples of 1024 (kilo-characters).

M

If specified, indicates that the length value is in multiples of 1024*1024 (mega-characters).

G

If specified, indicates that the length value is in multiples of 1024*1024*1024 (giga-characters).

Corresponding Compile-time Java Type

```
java.sql.Clob
```

JDBC Metadata Type ([java.sql.Types](#))

```
CLOB
```

Usage Notes

Use the `getClob` method on the `java.sql.ResultSet` to retrieve a CLOB handle to the underlying data.

There are a number of restrictions on using BLOB and CLOB / TEXT objects, which we refer to as LOB-types:

- » LOB-types cannot be compared for equality (=) and non-equality (!=, <>).

- » LOB-typed values cannot be ordered, so <, <=, >, >= tests are not supported.
- » LOB-types cannot be used in indexes or as primary key columns.
- » DISTINCT, GROUP BY, and ORDER BY clauses are also prohibited on LOB-types.
- » LOB-types cannot be involved in implicit casting as other base-types.

Example

```
CREATE TABLE myTable( txtCol TEXT(65535));
```

TIME

The TIME data type provides for storage of a time-of-day value.

Syntax

TIME

Corresponding Compile-time Java Type

`java.sql.Time`

JDBC Metadata Type (`java.sql.Types`)

TIME

Usage Notes

Here are several usage notes for the TIME data type:

- » timestamps cannot be mixed with one another in expressions except with a CAST.
- » Any value that is recognized by the `java.sql.Time` method is permitted in a column of the corresponding SQL date/time data type. Splice Machine supports the following formats for TIME:

```
hh:mm[:ss]
hh.mm[.ss]
hh[:mm] {AM | PM}
```

The first of the three formats above is the `java.sql.Time` format.

- » Hours may have one or two digits.
- » Minutes and seconds, if present, must have two digits.
- » Splice Machine also accepts strings in the locale specific date-time format, using the locale of the database server. If there is an ambiguity, the built-in formats above take precedence.

Please see [Working With Date and Time Values](#) for information about using simple arithmetic with TIME values.

Examples

```
VALUES TIME('15:09:02');  
VALUES '15:09:02';
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [*Working with Dates* in the *Developer's Guide*](#)

TIMESTAMP

The TIMESTAMP data type stores a combined [TIME](#) value that permits fractional seconds values of up to nine digits.

Syntax

`TIMESTAMP`

Corresponding Compile-time Java Type

`java.sql.Timestamp`

JDBC Metadata Type (`java.sql.Types`)

`TIMESTAMP`

About Timestamp Formats

Splice Machine uses the following Java date and time pattern letters to construct timestamps:

Pattern Letter	Description	Format(s)
y	year	yy or yyyy
M	month	MM
d	day in month	dd
h	hour (0-12)	hh
H	hour (0-23)	HH
m	minute in hour	mm
s	seconds	ss
S	tenths of seconds	sss (up to 6 decimal digits: SSSSSS)
z	time zone text	e.g. Pacific Standard time

Pattern Letter	Description	Format(s)
Z	time zone, time offset	e.g. -0800

The default timestamp format for Splice Machine imports is: yyyy-MM-dd HH:mm:ss, which uses a 24-hour clock, does not allow for decimal digits of seconds, and does not allow for time zone specification.

Please see [Working With Date and Time Values](#) for information about using simple arithmetic with `TIMESTAMP` values.

Examples

The following tables shows valid examples of timestamps and their corresponding format (parsing) patterns:

Timestamp value	Format Pattern	Notes
2013-03-23 09:45:00	yyyy-MM-dd HH:mm:ss	This is the default pattern.
2013-03-23 19:45:00.98-05	yyyy-MM-dd HH:mm:ss.SS	This pattern allows up to 2 decimal digits of seconds, and requires a time zone specification.
2013-03-23 09:45:00-07	yyyy-MM-dd HH:mm:ssZ	This pattern requires a time zone specification, but does not allow for decimal digits of seconds.
2013-03-23 19:45:00.98-0530	yyyy-MM-dd HH:mm:ss.SS	This pattern allows up to 2 decimal digits of seconds, and requires a time zone specification.
2013-03-23 19:45:00.123	yyyy-MM-dd HH:mm:ss.SSS	This pattern allows up to 3 decimal digits of seconds, but does not allow a time zone specification.
2013-03-23 19:45:00.12		Note that if your data specifies more than 3 decimal digits of seconds, an error occurs.
2013-03-23 19:45:00.1298	yyyy-MM-dd HH:mm:ss.SSSS	This pattern allows up to 4 decimal digits of seconds, but does not allow a time zone specification.

Usage Notes

Dates, times, and timestamps cannot be mixed with one another in expressions.

Splice Machine also accepts strings in the locale specific datetime format, using the locale of the database server. If there is an ambiguity, the built-in formats shown above take precedence.

At this time, dates in [TimeStamp](#) values only work correctly when limited to this range of date values: 1678-01-01 to 2261-12-31

See Also

- » [About Data Types](#)
- » [*Working with Dates*](#) in the *Developer's Guide*

VARCHAR

The VARCHAR data type provides for variable-length storage of strings.

Syntax

```
{ VARCHAR | CHAR VARYING | CHARACTER VARYING }(length)
```

length

An unsigned integer constant. The maximum length for a VARCHAR string is 32,672 characters.

Corresponding Compile-time Java Type

```
java.lang.String
```

JDBC Metadata Type (java.sql.Types)

VARCHAR

Example

```
VARCHAR(2048);
```

Usage Notes

Here are several notes for the VARCHAR data type:

- » Splice Machine does not pad a VARCHAR value whose length is less than specified.
- » Splice Machine truncates spaces from a string value when a length greater than the VARCHAR expected is provided. Characters other than spaces are not truncated, and instead cause an exception to be raised.
- » When comparison boolean operators are applied to VARCHARs, the lengths of the operands are not altered, and spaces at the end of the values are ignored.
- » When CHARs and VARCHARs are mixed in expressions, the shorter value is padded with spaces to the length of the longer value.
- » The type of a string constant is CHAR, not VARCHAR.

Statements

This section contains the reference documentation for the Splice Machine SQL Statements, in the following subsections:

- » Data Definition (DDL) - General Statements
- » Data Definition (DDL) - Create Statements
- » Data Definition (DDL) - Drop Statements
- » Data Manipulation (DML) Statements
- » [Session Control Statements](#)

Data Definition - General Statements

These are the data definition statements:

Statement	Description
ALTER TABLE	Add, deletes, or modifies columns in an existing table.
GRANT	Gives privileges to specific user(s) or role(s) to perform actions on database objects.
PIN TABLE	Caches a table in memory for improved performance.
RENAME COLUMN	Renames a column in a table.
RENAME INDEX	Renames an index in the current schema.
RENAME TABLE	Renames an existing table in a schema.
REVOKE	Revokes privileges for specific user(s) or role(s) to perform actions on database objects.
TRUNCATE TABLE	Resets a table to its initial empty state.
UNPIN TABLE	Unpins a pinned (cached) table.

Data Definition (DDL) - CREATE Statements

These are the create statements:

Statement	Description
CREATE EXTERNAL TABLE	Allows you to query data stored in a flat file as if that data were stored in a Splice Machine table.
CREATE FUNCTION	Creates Java functions that you can then use in expressions.
CREATE INDEX	Creates an index on a table.
CREATE PROCEDURE	Creates Java stored procedures, which you can then call using the Call Procedure statement.
CREATE ROLE	Creates SQL roles.
CREATE SCHEMA	Creates a schema.
CREATE SEQUENCE	Creates a sequence generator, which is a mechanism for generating exact numeric values, one at a time.
CREATE SYNONYM	Creates a synonym, which can provide an alternate name for a table or a view.
CREATE TABLE	Creates a new table.
CREATE TEMPORARY TABLE	Defines a temporary table for the current connection.
CREATE TRIGGER	Creates a trigger, which defines a set of actions that are executed when a database event occurs on a specified table
CREATE VIEW	Creates a view, which is a virtual table formed by a query.
DECLARE GLOBAL TEMPORARY TABLE	Defines a temporary table for the current connection.

Data Definition (DDL) - DROP Statements

These are the drop statements:

Statement	Description
DROP FUNCTION	Drops a function from a database.
DROP INDEX	Drops an index from a database.
DROP PROCEDURE	Drops a procedure from a database.
DROP ROLE	Drops a role from a database.

Statement	Description
DROP SCHEMA	Drops a schema from a database.
DROP SEQUENCE	Drops a sequence from a database.
DROP SYNONYM	Drops a synonym from a database.
DROP TABLE	Drops a table from a database.
DROP TRIGGER	Drops a trigger from a database.
DROP VIEW	Drops a view from a database.
DROP FUNCTION	Drops a function from a database.

Data Manipulation (DML) Statements

These are the data manipulation statements:

Statement	Description
CALL PROCEDURE	Calls a stored procedure.
DELETE	Deletes records from a table.
INSERT	Inserts records into a table.
SELECT	Selects records.
UPDATE TABLE	Updates values in a table.

Session Control Statements

These are the session control statements:

Statement	Description
SET ROLE	Sets the current role for the current SQL context of a session.
SET SCHEMA	Sets the default schema for a connection's session.



For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](#).

ALTER TABLE

The `ALTER TABLE` statement allows you to modify a table in a variety of ways, including adding and dropping columns and constraints from the table.



In this release, you **cannot** use `ALTER TABLE` to:

- » add a primary key
- » drop a foreign key constraint

Syntax

```
ALTER TABLE table-Name
{
  ADD COLUMN column-definition |
  ADD CONSTRAINT clause |
  DROP [ COLUMN ] column-name
  DROP { UNIQUE constraint-name |
         CHECK constraint-name
  }
  ALTER [ COLUMN ] column-alteration
}
```

column-definition

```
Simple-column-name [ DataType ]
[ Column-level-constraint ]*
[ [ WITH ] DEFAULT DefaultConstantExpression
  | generation-clause
]
```

The syntax for the *column-definition* for a new column is a subset of the syntax for a column in a [CREATE TABLE](#) statement.

The *DataType* can be omitted only if you specify a *generation-clause*. If you omit the *DataType*, the type of the generated column is the type of the *generation-clause*. If you specify both a *DataType* and a *generation-clause*, the type of the *generation-clause* must be assignable to *DataType*.

column-alteration

```
column-Name SET DATA TYPE VARCHAR(integer) |
column-name SET INCREMENT BY integer-constant |
column-name RESTART WITH integer-constant |
column-name [ NOT ] NULL |
column-name [ WITH | SET ] DEFAULT default-value |
column-name DROP DEFAULT
```

In the column-alteration, `SET INCREMENT BY` integer-constant specifies the interval between consecutive values of the identity column. The next value to be generated for the identity column will be determined from the last assigned value with the increment applied. The column must already be defined with the `IDENTITY` attribute.

`RESTART WITH` integer-constant specifies the next value to be generated for the identity column. `RESTART WITH` is useful for a table that has an identity column that was defined as `GENERATED BY DEFAULT` and that has a unique key defined on that identity column.

Because `GENERATED BY DEFAULT` allows both manual inserts and system generated values, it is possible that manually inserted values can conflict with system generated values. To work around such conflicts, use the `RESTART WITH` syntax to specify the next value that will be generated for the identity column.

Consider the following example, which involves a combination of automatically generated data and manually inserted data:

```
CREATE TABLE tauto(i INT GENERATED BY DEFAULT AS IDENTITY,
                    k INT)
CREATE UNIQUE INDEX tautoInd ON tauto(i)
INSERT INTO tauto(k) values 1,2;
```

The system will automatically generate values for the identity column. But now you need to manually insert some data into the identity column:

```
INSERT INTO tauto VALUES (3,3);
INSERT INTO tauto VALUES (4,4);
INSERT INTO tauto VALUES (5,5);
```

The identity column has used values 1 through 5 at this point. If you now want the system to generate a value, the system will generate a 3, which will result in a unique key exception because the value 3 has already been manually inserted. To compensate for the manual inserts, issue an `ALTER TABLE` statement for the identity column with `RESTART WITH 6`:

```
ALTER TABLE tauto ALTER COLUMN i RESTART WITH 6;
```

`ALTER TABLE` does not affect any view that references the table being altered. This includes views that have a wildcard asterisk (*) in their `SELECT` list. You must drop and re-create those views if you wish them to return the new columns.

To change a column constraint to `NOT NULL`, there has to be a valid value for the column.

Splice Machine raises an error if you try to change the `DataType` of a generated column to a type which is not assignable from the type of the `generation-clause`. Splice Machine also raises an error if you try to add a `DEFAULT` clause to a generated column.

Usage

The `ALTER TABLE` statement allows you to:

- » add a column to a table
- » add a constraint to a table

- » drop a column from a table
- » drop an existing constraint from a table*
- » increase the width of a VARCHAR column
- » change the increment value and start value of the identity column
- » change the nullability constraint for a column
- » change the default value for a column

Adding columns

The syntax for the [column-definition](#) for a new column is almost the same as for a column in a `CREATE TABLE` statement. This syntax allows a column constraint to be placed on the new column within the `ALTER TABLE ADD COLUMN` statement. However, a column with a `NOT NULL` constraint can be added to an existing table if you give a default value; otherwise, an exception is thrown when the `ALTER TABLE` statement is executed.

NOTE: If a table has an UPDATE trigger without an explicit column list, adding a column to that table in effect adds that column to the implicit update column list upon which the trigger is defined, and all references to transition variables are invalidated so that they pick up the new column.

Adding constraints

`ALTER TABLE ADD CONSTRAINT` adds a table-level constraint to an existing table.



The `ALTER TABLE ADD CONSTRAINT` statement is not currently taking currently running transactions into account, and thus can fail to add the constraint. This issue will be resolved in a future release.

You can reliably add constraints when using the `CREATE TABLE` statement.

The following limitations exist on adding a constraint to an existing table:

- » When adding a check constraint to an existing table, Splice Machine checks the table to make sure existing rows satisfy the constraint. If any row is invalid, Splice Machine throws a statement exception and the constraint is not added.

For information on the syntax of constraints, see [CONSTRAINT](#) clause. Use the syntax for table-level constraint when adding a constraint with the `ADD TABLE ADD CONSTRAINT` syntax.

Dropping columns

`ALTER TABLE DROP COLUMN` allows you to drop a column from a table.

The keyword `COLUMN` is optional.

You may not drop the last (only) column in a table.

Modifying columns

The [column-alteration](#) allows you to alter the named column in the following ways:

- » Increasing the width of an existing VARCHAR column. CHARACTER VARYING or CHAR VARYING can be used as synonyms for the VARCHAR keyword.

To increase the width of a column of these types, specify the data type and new size after the column name.

You are not allowed to decrease the width or to change the data type. You are not allowed to increase the width of a column that is part of a primary or unique key referenced by a foreign key constraint or that is part of a foreign key constraint.

- » Specifying the interval between consecutive values of the identity column.

To set an interval between consecutive values of the identity column, specify the integer-constant. You must previously define the column with the IDENTITY attribute (SQLSTATE 42837). If there are existing rows in the table, the values in the column for which the SET INCREMENT default was added do not change.

- » Modifying the nullability constraint of a column.

You can add the NOT NULL constraint to an existing column; however, you cannot do so if there are NULL values for the column in the table.

You can remove the NOT NULL constraint from an existing column; however, you cannot do so if the column is used in a PRIMARY KEY constraint.

- » Changing the default value for a column.

You can use DEFAULT default-value to change a column default. To disable a previously set default, use DROP DEFAULT (alternatively, you can specify NULL as the default-value).

Setting defaults

You can specify a default value for a new column. A default value is the value that is inserted into a column if no other value is specified. If not explicitly specified, the default value of a column is NULL. If you add a default to a new column, existing rows in the table gain the default value in the new column.

For more information about defaults, see [CREATE TABLE](#) statement.

An ALTER TABLE statement causes all statements that are dependent on the table being altered to be recompiled before their next execution.

Examples

This section provides examples of using the ALTER TABLE statement.

Example 1: Adding Columns to a Table

In this example, we create a new table, and then use `ALTER TABLE` statements to add three columns that we have decided to include:

```
splice> CREATE TABLE PlayerTrades (
    ID INT NOT NULL,
    PlayerName VARCHAR(32),
    Position CHAR(2),
    OldTeam VARCHAR(32),
    NewTeam VARCHAR(32) );
0 rows inserted/updated/deleted

splice> ALTER TABLE PlayerTrades ADD COLUMN Updated TIMESTAMP;
0 rows inserted/updated/deleted

splice> ALTER TABLE PlayerTrades ADD COLUMN TradeDate DATE;
0 rows inserted/updated/deleted

splice> ALTER TABLE PlayerTrades ADD COLUMN Years INT;
0 rows inserted/updated/deleted

splice> INSERT INTO PlayerTrades VALUES( 1, 'Greinke', 'SP', 'Dodgers', 'Giants', CURRENT_TIMESTAMP, CURRENT_DATE);
1 row inserted/updated/deleted

splice> DESCRIBE PlayerTrades;
COLUMN_NAME | TYPE_NAME | DEC&| NUM& | COLUM& | COLUMN_DEF | CHAR_OCTE& | IS_NULL&
-----
ID          | INTEGER   | 0   | 10  | 10   | NULL      | NULL      | NO
PLAYERNAME  | VARCHAR   | NULL| NULL | 32   | NULL      | 64        | YES
POSITION    | CHAR      | NULL| NULL | 2    | NULL      | 4         | YES
OLDTEAM     | VARCHAR   | NULL| NULL | 32   | NULL      | 64        | YES
NEWTEAM     | VARCHAR   | NULL| NULL | 32   | NULL      | 64        | YES
UPDATED     | TIMESTAMP | 9   | 10  | 29   | NULL      | NULL      | YES
TRADEDATE   | DATE      | 0   | 10  | 10   | NULL      | NULL      | YES
YEARS       | INTEGER   | 0   | 10  | 10   | NULL      | NULL      | YES

8 rows selected
```

Example 2: Altering Columns

In this example, we use `ALTER TABLE` to alter columns in various ways:

- » specify that the `Updated` column cannot be `NULL`
- » set the default value for `Years` to 3
- » set the default value for `NewTeam` to 'Giants'

```

splice> ALTER TABLE PlayerTrades ALTER COLUMN Updated NOT NULL;
0 rows inserted/updated/deleted

splice> ALTER TABLE PlayerTrades ALTER COLUMN Years DEFAULT 3;
0 rows inserted/updated/deleted

splice> ALTER TABLE PlayerTrades ALTER COLUMN NewTeam DEFAULT 'Giants';
0 rows inserted/updated/deleted

splice> DESCRIBE PlayerTrades;
COLUMN_NAME | TYPE_NAME | DEC& | NUM& | COLUM& | COLUMN_DEF | CHAR_OCTE& | IS_NULL&
-----
ID          | INTEGER    | 0      | 10     | 10      | NULL      | NULL      | NO
PLAYERNAME   | VARCHAR    | NULL   | NULL   | 32      | NULL      | 64        | YES
POSITION     | CHAR       | NULL   | NULL   | 2       | NULL      | 4         | YES
OLDTEAM      | VARCHAR    | NULL   | NULL   | 32      | NULL      | 64        | YES
NEWTEAM      | VARCHAR    | NULL   | NULL   | 32      | 'Giants'  | 64        | YES
UPDATED      | TIMESTAMP  | 9      | 10     | 29      | NULL      | NULL      | NO
TRADEDATE    | DATE       | 0      | 10     | 10      | NULL      | NULL      | YES
YEARS        | INTEGER    | 0      | 10     | 10      | 3         | NULL      | YES

7 rows selected

```

Example 3: Dropping a column

This example drops the Years column from our table, and then drops the default associated with NewTeam:

```

splice> ALTER TABLE PlayerTrades DROP COLUMN Years;
0 rows inserted/updated/deleted

splice> ALTER TABLE PlayerTrades ALTER COLUMN NewTeam DROP DEFAULT;
0 rows inserted/updated/deleted

splice> DESCRIBE PlayerTrades;
COLUMN_NAME | TYPE_NAME | DEC& | NUM& | COLUM& | COLUMN_DEF | CHAR_OCTE& | IS_NULL&
-----
ID          | INTEGER    | 0      | 10     | 10      | NULL      | NULL      | NO
PLAYERNAME   | VARCHAR    | NULL   | NULL   | 32      | NULL      | 64        | YES
POSITION     | CHAR       | NULL   | NULL   | 2       | NULL      | 4         | YES
OLDTEAM      | VARCHAR    | NULL   | NULL   | 32      | NULL      | 64        | YES
NEWTEAM      | VARCHAR    | NULL   | NULL   | 32      | NULL      | 64        | YES
UPDATED      | TIMESTAMP  | 9      | 10     | 29      | NULL      | NULL      | NO
TRADEDATE    | DATE       | 0      | 10     | 10      | NULL      | NULL      | YES

7 rows selected

```

Example 4: Changing Varchar Column Width

This example changes the width of one of our VARCHAR columns:

```
splice> ALTER TABLE PlayerTrades ALTER COLUMN PlayerName SET DATA TYPE VARCHAR(40);
0 rows inserted/updated/deleted

splice> DESCRIBE PlayerTrades;
COLUMN_NAME | TYPE_NAME | DEC& | NUM& | COLUM& | COLUMN_DEF | CHAR_OCTE& | IS_NULL&
-----
ID          | INTEGER   | 0    | 10   | 10    | NULL     | NULL      | NO
PLAYERNAME  | VARCHAR   | NULL | NULL | 40    | NULL     | 80        | YES
POSITION    | CHAR      | NULL | NULL | 2     | NULL     | 4         | YES
OLDTEAM     | VARCHAR   | NULL | NULL | 32    | NULL     | 64        | YES
NEWTEAM     | VARCHAR   | NULL | NULL | 32    | NULL     | 64        | YES
UPDATED     | TIMESTAMP | 9    | 10   | 29    | NULL     | NULL      | NO
TRADEDATE   | DATE      | 0    | 10   | 10    | NULL     | NULL      | YES
-----
```

7 rows selected

Example 5: Changing Increment Value

This example shows creating a table with an identity column, and then changing the increment for that column:

```
splice> CREATE TABLE NewPlayers(
    newID INT NOT NULL GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    PlayerName);
0 rows inserted/updated/deleted

splice> ALTER TABLE NewPlayers ALTER COLUMN newID SET INCREMENT BY 10;
0 rows inserted/updated/deleted

splice> INSERT INTO NewPlayers(PlayerName) ('Greinke'), ('Cespedes');
2 rows inserted/updated/deleted

splice> SELECT * FROM NewPlayers;
NEWID      | PLAYERNAME
-----
1          | Greinke
11         | Cespedes
-----
```

2 rows selected

See Also

- » [CONSTRAINT clause](#)
- » [CREATE TABLE statement](#)
- » [Foreign Keys](#) in the *Developer's Guide*.
- » [Triggers](#) in the *Developer's Guide*.

CALL (Procedure)

The CALL (PROCEDURE) statement is used to call stored procedures.

When you call a stored procedure, the default schema and role are the same as were in effect when the procedure was created.

Syntax

```
CALL procedure-Name (
    [ expression [, expression]* ]
)
```

procedure-Name

The name of the procedure that you are calling.

expression(s)

Arguments passed to the procedure.

Example

The following example depends on a fictionalized java class. For functional examples of using CREATE PROCEDURE, please see the [Using Functions and Stored Procedures](#) section in our *Developer's Guide*.

```
splice> CREATE PROCEDURE SALES.TOTAL_REVENUE(IN S_MONTH INTEGER,
    IN S_YEAR INTEGER, OUT TOTAL DECIMAL(10,2))
PARAMETER STYLE JAVA
READS SQL DATA LANGUAGE JAVA EXTERNAL NAME
    'com.example.sales.calculateRevenueByMonth';
splice> CALL SALES.TOTAL_REVENUE(?, ?, ?);
```

See Also

» [CREATE PROCEDURE statement](#)

column-definition

```
Simple-column-Name
[ DataType ]
[ Column-level-constraint ]*
[ [ WITH ] DEFAULT DefaultConstantExpression
  | generated-column-spec
  | generation-clause
]
[ Column-level-constraint ]*
```

DataType

Must be specified unless you specify a generation-clause, in which case the type of the generated column is that of the generation-clause.

If you specify both a *DataType* and a *generation-clause*, the type of the *generation-clause* must be assignable to *DataType*.

Column-level-constraint

See the [CONSTRAINT](#) clause documentation.

DefaultConstantExpression

For the definition of a default value, a *DefaultConstantExpression* is an expression that does not refer to any table. It can include constants, date-time special registers, current schemas, and null:

```
DefaultConstantExpression:
  NULL
  | CURRENT { SCHEMA | SQLID }
  | DATE
  | TIME
  | TIMESTAMP
  | CURRENT_DATE | CURRENT_DATE
  | CURRENT_TIME | CURRENT_TIME
  | CURRENT_TIMESTAMP | CURRENT_TIMESTAMP
  | literal
```

The values in a *DefaultConstantExpression* must be compatible in type with the column, but a *DefaultConstantExpression* has the following additional type restrictions:

- » If you specify CURRENT SCHEMA or CURRENT SQLID, the column must be a character column whose length is at least 128.
- » If the column is an integer type, the default value must be an integer literal.
- » If the column is a decimal type, the scale and precision of the default value must be within those of the column.

Example

This example creates a table and uses two column definitions.

```
splice> CREATE TABLE myTable (ID INT NOT NULL, NAME VARCHAR(32) );
0 rows inserted/updated/deleted
```

See Also

- » [CONSTRAINT clause](#)

CREATE EXTERNAL TABLE

A `CREATE EXTERNAL TABLE` statement creates a table in Splice Machine that you can use to query data that is stored externally in a flat file, such as a file in Parquet, ORC, or plain text format. External tables are largely used as a convenient means of moving data into and out of your database.

You can query external tables just as you would a regular Splice Machine table; however, you cannot perform any DML operations on an external table, once it has been created. That also means that you cannot create an index on an external table.

If the schema of the external file that you are querying is modified outside of Splice, you need to manually refresh the Splice Machine table by calling the `REFRESH EXTERNAL TABLE` built-in system procedure.

If a qualified table name is specified, the schema name cannot begin with `SYS`.

Syntax

```
CREATE EXTERNAL TABLE table-Name
{
  ( column-definition* )
  [ COMPRESSED WITH compression-format ]
  [ PARTITIONED BY (column-name ) ]}
  [ ROW FORMAT DELIMITED
    [ FIELDS TERMINATED BY char [ESCAPED BY char] ]
    [ LINES TERMINATED BY char ]
  ]
  STORED AS file-format LOCATION location
}
```

table-Name

The name to assign to the new table.

compression-format

The compression algorithm used to compress the flat file source of this external table. You can specify one of the following values:

- » ZLIB
- » SNAPPY

If you don't specify a compression format, the default is uncompressed. You cannot specify a *compression-format* when using the `TEXTFILE` *file-format*; doing so generates an error.

column-definition

A column definition.

The maximum number of columns allowed in a table is 100000.

column-name

The name of a column.

char

A single character used as a delimiter or escape character. Enclose this character in single quotes; for example, ‘,’.

To specify a special character that includes the backslash character, you must escape the backslash character itself. For example:

- » \\\ to indicate a backslash character
- » \n to indicate a newline character
- » \t to indicate a tab character

file-format

The format of the flat file source of this external table. This is currently one of these values:

- » ORC is a columnar storage format
- » PARQUET is a columnar storage format
- » Avro is a data serialization system
- » TEXTFILE is a plain text file

location

The location at which the file is stored.

Usage Notes

Here are some notes about using external tables:

- » If the data types in the table schema you specify do not match the schema of the external file, an error occurs and the table is not created.
- » You cannot define indexes or constraints on external tables
- » The ROW FORMAT parameter is only applicable to plain text (TextFile) not supported for columnar storage format files (ORC or PARQUET files)
- » If you specify the location of a non-existent file when you create an external table, Splice Machine automatically creates an external file at that location.
- » AVRO external tables do not currently work with compressed files; any compression format you specify will be ignored.
- » Splice Machine isn't able to know when the schema of the file represented by an external table is updated; when this occurs, you need to update the external table in Splice Machine by calling the [SYSCS_UTIL.SYSCS_REFRESH_EXTERNAL_TABLE](#) built-in system procedure.
- » You cannot specify a *compression-format* when using the TEXTFILE *file-format*; doing so generates an error.

Examples

This section presents examples of the CREATE EXTERNAL TABLE statement.

This example creates an external table for a PARQUET file:

```
splice> CREATE EXTERNAL TABLE myParquetTable(
    col1 INT, col2 VARCHAR(24))
    PARTITIONED BY (col1)
    STORED AS PARQUET
    LOCATION '/users/myName/myParquetFile'
);
0 rows inserted/updated/deleted
```

This example creates an external table for an AVRO file:

```
splice> CREATE EXTERNAL TABLE myAvroTable(
    col1 INT, col2 VARCHAR(24))
    PARTITIONED BY (col1)
    STORED AS AVRO
    LOCATION '/users/myName/myAvroFile'
);
0 rows inserted/updated/deleted
```

This example creates an external table for an ORC file and inserts data into it:

```
splice> CREATE EXTERNAL TABLE myOrcTable(
    col1 INT, col2 VARCHAR(24))
    PARTITIONED BY (col1)
    STORED AS ORC
    LOCATION '/users/myName/myOrcFile'
);
0 rows inserted/updated/deleted
splice> INSERT INTO myOrcTable VALUES (1, 'One'), (2, 'Two'), (3, 'Three');
3 rows inserted/updated/deleted
splice> SELECT * FROM myOrcTable;
COL1      |COL2-----
3        |Three
2        |Two
1        |One
```

This example creates an external table for a plain text file:

```
splice> CREATE EXTERNAL TABLE myTextTable(
          col1 INT, col2 VARCHAR(24))
          PARTITIONED BY (col1)
          ROW FORMAT DELIMITED FIELDS
          TERMINATED BY ','
          ESCAPED BY '\\'
          LINES TERMINATED BY '\\n'
          STORED AS TEXTFILE
          LOCATION '/users/myName/myTextFile'
      );
0 rows inserted/updated/deleted
```

This example creates an external table for a PARQUET file that was compressed with Snappy compression:

```
splice> CREATE EXTERNAL TABLE mySnappyParquetTable(
          col1 INT, col2 VARCHAR(24))
          COMPRESSED WITH SNAPPY
          PARTITIONED BY (col1)
          STORED AS PARQUET
          LOCATION '/users/myName/mySnappyParquetFile'
);
0 rows inserted/updated/deleted
```

See Also

- » [CREATE TABLE](#)
- » [PIN TABLE](#)
- » [DROP TABLE](#)
- » [REFRESH EXTERNAL TABLE](#)
- » [Foreign Keys](#)
- » [Triggers](#)

CREATE FUNCTION

The `CREATE FUNCTION` statement allows you to create Java functions, which you can then use in expressions.

For details on how Splice Machine matches functions to Java methods, see [Argument matching](#).

Syntax

```
CREATE FUNCTION functionName (
    [ functionParameter ]
    [, functionParameter] ] *
)
RETURNS returnType [ functionElement ] *
```

functionName

SQL Identifier

If `schemaName` is not provided, then the current schema is the default schema. If a qualified procedure name is specified, the schema name cannot begin with `SYS`.

functionParameter

[*parameterName*] *DataType*

parameterName is an identifier that must be unique within the function's parameter names.

NOTE: Data-types such as `BLOB`, `CLOB`, `LONG VARCHAR` are not allowed as parameters in a `CREATE FUNCTION` statement.

returnDataType

DataType |
TableType

functionElement

See the description of [Function Elements](#) in the next section.

TableType

```
TABLE( ColumnElement [ ,ColumnElement]*) )
```

ColumnElement

A *SQL Identifier*.

Table functions return `TableType` results. Currently, only Splice Machine-style table functions are supported, which are functions that return JDBC `ResultSets`.

When values are extracted from a `ResultSet`, the data types of the values are coerced to match the data types declared in the `CREATE FUNCTION` statement. Here are a few coercion rules you should know about:

- » values that are too long are truncated to the maximum declared length
- » if a string value is returned in the `ResultSet` for a column of type `CHAR`, and the string is shorter than the column length, the string is padded with spaces

Function Elements

```
{
  LANGUAGE { JAVA }
  DeterministicCharacteristic
  EXTERNAL NAME javaMethodName
  PARAMETER STYLE parameterStyle
  sqlStatementType
  nullInputAction
}
```

The function elements may appear in any order, but each type of element can only appear once.

These function elements are required:

- » `LANGUAGE`
- » `EXTERNAL NAME`
- » `PARAMETER STYLE`

`LANGUAGE`

Only `JAVA` is accepted at this time. Splice Machine will call the function as a public static method in a Java class.

`DeterministicCharacteristic`

`DETERMINISTIC | NOT DETERMINISTIC`

The default value is `NOT DETERMINISTIC`.

Specifying `DETERMINISTIC` indicates that the function always returns the same result, given the same input values. This allows Splice Machine to call the function with greater efficiency; however, specifying this for a function that is actually non-deterministic will have the opposite effect – efficiency of calls to the function will be reduced.

`javaMethodName`

`class_name.method_name`

This is the name of the Java method to call when this function executes.

parameterStyle

JAVA | DERBY_JDBC_RESULT_SET

Only use DERBY_JDBC_RESULT_SET if this is a Splice Machine-style table function that returns a [TableType](#) result, and is mapped to a Java method that returns a JDBC *ResultSet*.

Otherwise, use JAVA-style parameters, which means that a parameter-passing convention is used that conforms to the Java language and SQL Routines specification. INOUT and OUT parameters are passed as single entry arrays to facilitate returning values. Result sets can be returned through additional parameters to the Java method of type `java.sql.ResultSet[]` that are passed single entry arrays.

Splice Machine does not support long column types such as LONG VARCHAR or BLOB; an error will occur if you try to use one of these long column types.

*sqlStatementType**CONTAINS SQL*

Indicates that SQL statements that neither read nor modify SQL data can be executed by the function. Statements that are not supported in any function return a different error.

NO SQL

Indicates that the function cannot execute any SQL statements

READS SQL DATA

Indicates that some SQL statements that do not modify SQL data can be included in the function. Statements that are not supported in any stored function return a different error. This is the default value.

*nullInputAction**RETURNS NULL ON NULL INPUT*

If any input argument is null, the function is not invoked, and the result is null.

CALLED ON NULL INPUT

This is the default value.

The function is invoked even if all input arguments are null, which means that the invoked function must test for null argument values. The result may be null or not null.

Example of declaring a scalar function

For more complete examples of using `CREATE FUNCTION`, please see the [Using Functions and Stored Procedures](#) section of our *Developer's Guide*.

```
splice> CREATE FUNCTION TO_DEGREES( RADIANS DOUBLE )
  RETURNS DOUBLE
  PARAMETER STYLE JAVA
  NO SQL LANGUAGE JAVA
  EXTERNAL NAME 'java.lang.Math.toDegrees';

0 rows inserted/updated/deleted
```

Example of declaring a table function

This example reads data from a mySql database and inserts it into a Splice Machine database.

We first implement a class that contains a public static method that connects to an external (foreign) database, uses a prepared statement to pull results from it, and returns those results as a JDBC ResultSet:

```
package splicemachine.example.vti;
import java.sql.*;
public class EmployeeTable{
    public static ResultSet read() throws SQLException {
        Connection conn DriverManager.getConnection(
            "jdbc:mysql://localhost/hr?user=myName&password=myPswd" );
        PreparedStatement ps = conn.prepareStatement(
            "SELECT * FROM hrSchema.EmployeeTable" );
        return ps.executeQuery();
    }
}
```

Next we use the [CREATE FUNCTION](#) .statement to declare a table function to read data from our external database and insert it into our Splice Machine database:

```
CREATE FUNCTION externalEmployees()
RETURNS TABLE
(
  employeeId      INT,
  lastName        VARCHAR( 50 ),
  firstName        VARCHAR( 50 ),
  birthday        DATE
)
LANGUAGE JAVA
PARAMETER STYLE SPLICE_JDBC_RESULT_SET    READS SQL DATA    EXTERNAL NAME 'com.splicemachine.example.vti.readEmployees';
```

Now we're ready to invoke our table function to read data from the external database and insert it into a table in our Splice Machine database.

To invoke a table function, you must wrap it in a TABLE constructor in the FROM list of a query. For example, we could insert employee data from that database into a table named employees in our Splice Machine database:

```
INSERT INTO employees
  SELECT myExtTbl.*
    FROM TABLE (externalEmployees() ) myExtTbl;
```

NOTE: You **MUST** specify the table alias when using a virtual table; for example, myExtTbl in the above example.

See Also

- » [CREATE_PROCEDURE statement](#)
- » [CURRENT_USER function](#)
- » [Data Types](#)
- » [DROP FUNCTION statement](#)
- » [Schema Name](#)
- » [SQL Identifier](#)
- » [SESSION_USER function](#)
- » [USER function](#)

CREATE INDEX

A CREATE INDEX statement creates an index on a table. Indexes can be on one or more columns in the table.

Syntax

```
CREATE [UNIQUE] INDEX indexName
  ON tableName (
    simpleColumnName
    [ ASC | DESC ]
    [ , simpleColumnName [ ASC | DESC ] ] *
  )
  [ SPLITKEYS splitKeyInfo HFILE hfileLocation ]
```

indexName

An identifier, the length of which cannot exceed 128 characters.

tableName

A table name, which can optionally be qualified by a schema name.

simpleColumnName

A simple column name.

You cannot use the same column name more than once in a single CREATE INDEX statement. Note, however, that a single column name can be used in multiple indexes.

splitKeyInfo

```
AUTO |
{ LOCATION filePath
  [ colDelimiter ]
  [ charDelimiter ]
  [ timestampFormat ]
  [ dateFormat ]
  [ timeFormat ]
}
```

Use the optional SPLITKEYS section to create indexes using HFile Bulk Loading, which is described in the [Using Bulk Hfile Indexing](#) section, below. Using bulk HFiles improves performance for large datasets, and is related to our Bulk HFile Import procedure.

You can specify AUTO to have Splice Machine scan the data and determine the splits automatically. Or you can specify your own split keys in a CSV file; if you're using a CSV file, you can optionally include delimiter and format specifications, as described in the following parameter definitions. Each parameter name links to a fuller description of the possible parameter values, which are the similar to those used in our Import Parameters Tutorial.

colDelimiter

The character used to separate columns. You don't need to specify this if using the comma (,) character as your delimiter.

charDelimiter

The character is used to delimit strings in the imported data. You don't need to specify this if using the double-quote (\ ") character as your delimiter.

timeStampFormat

The format of timestamps stored in the file. You don't need to specify this if no time columns in the file, or if the format of any timestamps in the file match the Java.sql.Timestamp default format, which is: "yyyy-MM-dd HH:mm:ss".

dateFormat

The format of datestamps stored in the file. You don't need to specify this if there are no date columns in the file, or if the format of any dates in the file match the pattern: "yyyy-MM-dd".

timeFormat

The format of time values stored in the file. You can set this to null if there are no time columns in the file, or if the format of any times in the file match pattern: "HH:mm:ss".

hFileLocation

The location (full path) in which the temporary HFiles will be created. These files will automatically be deleted after the index creation process completes. This parameter is required when specifying split keys.

Usage

Splice Machine can use indexes to improve the performance of data manipulation statements. In addition, UNIQUE indexes provide a form of data integrity checking.

Index names are unique within a schema. (Some database systems allow different tables in a single schema to have indexes of the same name, but Splice Machine does not.) Both index and table are assumed to be in the same schema if a schema name is specified for one of the names, but not the other. If schema names are specified for both index and table, an exception will be thrown if the schema names are not the same. If no schema name is specified for either table or index, the current schema is used.

You cannot create an index that has the same index columns as an existing index; if you attempt to do so, Splice Machine issues a warning and does not create the index, as you can see in this example:

```
splice> CREATE INDEX idx1 ON myTable(id, eventType);
0 rows inserted/updated/deleted
splice> CREATE INDEX idx2 ON myTable(id, eventType);
WARNING 01504: The new index is a duplicate of an existing index: idx1.
splice> DROP INDEX idx2;
ERROR 42X65: Index 'idx2' does not exist.
```

By default, Splice Machine uses the ascending order of each column to create the index. Specifying ASC after the column name does not alter the default behavior. The DESC keyword after the column name causes Splice Machine to use descending order for the column to create the index. Using the descending order for a column can help improve the performance of queries that require the results in mixed sort order or descending order and for queries that select the minimum or maximum value of an indexed column.

If a qualified index name is specified, the schema name cannot begin with SYS.

Using Bulk HFiles to Create an Index

Bulk HFile indexing improves performance when indexing very large datasets. The table you're indexing is temporarily converted into HFiles to take advantage of HBase bulk loading; once the indexing operation is complete, the temporary HFiles are automatically deleted. This is very similar to using HFile Bulk Loading for importing large datasets, which is described in our Bulk HFile Import Tutorial.

You can have Splice Machine automatically determine the splits by scanning the data, or you can define the split keys in a CSV file. In the following example, we use our understanding of the Orders table to first create a CSV file named `ordersKey.csv` that contains the split keys we want, and then use the following `CREATE INDEX` statement to create the index:

```
CREATE INDEX o_Cust_Idx on Orders(
    o_custKey,
    o_orderKey
)
SPLITKEYS LOCATION '/tmp/ordersKey.csv'
    COLDELIMITER '|'
    HFILE LOCATION '/tmp/Hfiles';
```

The `/tmp/ordersKey.csv` file specifies the index keys; it uses the `|` character as a column delimiter. The temporary HFiles are created in the `/tmp/Hfiles` directory.

Indexes and constraints

Unique and primary key constraints generate indexes that enforce or “back” the constraint (and are thus sometimes called *backing indexes*). If a column or set of columns has a `UNIQUE` or `PRIMARY KEY` constraint on it, you can not create an index on those columns.

Splice Machine has already created it for you with a system-generated name. System-generated names for indexes that back up constraints are easy to find by querying the system tables if you name your constraint. Adding a `PRIMARY KEY` or `UNIQUE` constraint when an existing `UNIQUE` index exists on the same set of columns will result in two physical indexes on the table for the same set of columns. One index is the original `UNIQUE` index and one is the backing index for the new constraint.

Statement Dependency System

Prepared statements that involve `SELECT`, `INSERT`, `UPDATE`, and `DELETE` on the table referenced by the `CREATE INDEX` statement are invalidated when the index is created.

Example

```
splice> CREATE TABLE myTable (ID INT NOT NULL, NAME VARCHAR(32) NOT NULL );
0 rows inserted/updated/deleted

splice> CREATE INDEX myIdx ON myTable(ID);
0 rows inserted/updated/deleted
```

See Also

- » [DELETE statement](#)
- » [DROP INDEX statement](#)
- » [INSERT statement](#)
- » [SELECT statement](#)
- » [UPDATE statement](#)

CREATE PROCEDURE

The `CREATE PROCEDURE` statement allows you to create Java procedures, which you can then call using the `CALL PROCEDURE` statement.

For details on how Splice Machine matches procedures to Java methods, see Argument matching.

Syntax

```
CREATE PROCEDURE procedureName (
    [ procedureParameter
    [, procedureParameter] ] *
)
[ ProcedureElement ] *
```

procedureName

[*SQL Identifier*]

If `schemaName` is not provided, then the current schema is the default schema. If a qualified procedure name is specified, the schema name cannot begin with `SYS`.

procedureParameter

[{ `IN` | `OUT` | `INOUT` }] [*parameterName*] *DataType*

parameterName is an identifier that must be unique within the procedure's parameter names.

By default, parameters are `IN` parameters unless you specify otherwise.

Data-types such as `BLOB`, `CLOB`, `LONG VARCHAR` are not allowed as parameters in a `CREATE PROCEDURE` statement.

NOTE: Also: At this time, Splice Machine will return only one `ResultSet` from a stored procedure.

procedureElement

See the description of [procedure Elements](#) in the next section.

Procedure Elements

```
{
  LANGUAGE { JAVA }
  DeterministicCharacteristic
  EXTERNAL NAME javaMethodName
  PARAMETER STYLE parameterStyle
  sqlStatementType
}
```

The procedure elements may appear in any order, but each type of element can only appear once. These procedure elements are required:

- » *LANGUAGE*
- » *EXTERNAL NAME*
- » *PARAMETER STYLE*

LANGUAGE

Only JAVA is accepted at this time. Splice Machine will call the procedure as a public static method in a Java class.

DeterministicCharacteristic

```
DETERMINISTIC | NOT DETERMINISTIC
```

The default value is NOT DETERMINISTIC.

Specifying DETERMINISTIC indicates that the procedure always returns the same result, given the same input values. This allows Splice Machine to call the procedure with greater efficiency; however, specifying this for a procedure that is actually non-deterministic will have the opposite effect – efficiency of calls to the procedure will be reduced.

javaMethodName

```
class_name.method_name
```

This is the name of the Java method to call when this procedure executes.

parameterStyle

```
JAVA
```

Stored procedures use a parameter-passing convention that conforms to the Java language and SQL Routines specification. INOUT and OUT parameters are passed as single entry arrays to facilitate returning values. Result sets can be returned through additional parameters to the Java method of type `java.sql.ResultSet[]` that are passed single entry arrays.

Splice Machine does not support long column types such as LONG VARCHAR or BLOB; an error will occur if you try to use one of these long column types.

sqlStatementType

CONTAINS SQL

Indicates that SQL statements that neither read nor modify SQL data can be executed by the procedure.

NO SQL

Indicates that the procedure cannot execute any SQL statements

READS SQL DATA

Indicates that some SQL statements that do not modify SQL data can be included in the procedure. This is the default value.

MODIFIES SQL DATA

Indicates that the procedure can execute any SQL statement.

Example

The following example depends on a fictionalized java class. For functional examples of using CREATE PROCEDURE, please see the [Using Functions and Stored Procedures](#) section of our *Developer's Guide*.

```
splice> CREATE PROCEDURE SALES.TOTAL_REVENUE (
    IN S_MONTH INTEGER,
    IN S_YEAR INTEGER, OUT TOTAL DECIMAL(10,2))
PARAMETER STYLE JAVA
READS SQL DATA LANGUAGE
JAVA EXTERNAL NAME 'com.example.sales.calculateRevenueByMonth';
0 rows inserted/updated/deleted
```

See Also

- » Argument matching
- » [CREATE_FUNCTION](#) statement
- » [CURRENT_USER](#) function
- » Data Types
- » Schema Name
- » SQL Identifier
- » [SESSION_USER](#) function
- » [USER](#) function

CREATE ROLE

The CREATE ROLE statement allows you to create an SQL role. Only the database owner can create a role.

Syntax

```
CREATE ROLE roleName
```

roleName

The name of an SQL role.

Using

Before you issue a CREATE ROLE statement, verify that the *derby.database.sqlAuthorization* property is set to TRUE. The *derby.database.sqlAuthorization* property enables SQL authorization mode.

You cannot create a role name if there is already a user by that name. An attempt to create a role name that conflicts with an existing user name raises the *SQLException X0Y68*. If user names are not controlled by the database owner (or administrator), it may be a good idea to use a naming convention for roles to reduce the possibility of collision with user names.

Splice Machine tries to avoid name collision between user names and role names, but this is not always possible, because Splice Machine has a pluggable authorization architecture. For example, an externally defined user may exist who has never yet connected to the database, created any schema objects, or been granted any privileges. If Splice Machine knows about a user name, it will forbid creating a role with that name. Correspondingly, a user who has the same name as a role will not be allowed to connect. Splice Machine built-in users are checked for collision when a role is created.

A role name cannot start with the prefix SYS (after case normalization). The purpose of this restriction is to reserve a name space for system-defined roles at a later point. Use of the prefix SYS raises the *SQLException 4293A*.

You cannot create a role with the name PUBLIC (after case normalization). PUBLIC is a reserved authorization identifier. An attempt to create a role with the name PUBLIC raises *SQLException 4251B*.

Examples

Creating a Role

Here's a simple example of creating a role:

```
splice> CREATE ROLE statsEditor_role;
0 rows inserted/updated/deleted
```

Examples of Invalid Role Names

Here are several examples of attempts to create a role using names that are reserved and cannot be used as role names. Each of these generates an error:

```
splice> CREATE ROLE public;
splice> CREATE ROLE "PUBLIC";
splice> CREATE ROLE sysrole;
```

See Also

- » [DROP_ROLE statement](#)
- » [GRANT statement](#)
- » [REVOKE statement](#)
- » [RoleName](#)
- » [SET ROLE statement](#)
- » [SYSROLES system table](#)

CREATE SCHEMA

The `CREATE SCHEMA` statement allows you to create a database schema, which is a way to logically group objects in a single collection and provide a unique name-space for those objects.

Syntax

```
CREATE SCHEMA {  
    [ schemaName ]  
}
```

The `CREATE SCHEMA` statement is used to create a schema. A schema name cannot exceed 128 characters. Schema names must be unique within the database.

A schema name cannot start with the prefix `SYS` (after case normalization). Use of the prefix `SYS` raises a `SQLException`.

CREATE SCHEMA examples

To create a schema for airline-related tables, use the following syntax:

```
splice> CREATE SCHEMA FLIGHTS;  
0 rows inserted/updated/deleted
```

To create a schema employee-related tables, use the following syntax:

```
splice> CREATE SCHEMA EMP;  
0 rows inserted/updated/deleted
```

To create a table called `availability` in the `EMP` and `FLIGHTS` schemas, use the following syntax:

```
splice> CREATE TABLE Flights.Availability(
    Flight_ID CHAR(6) NOT NULL,
    Segment_Number INT NOT NULL,
    Flight_Date DATE NOT NULL,
    Economy_Seats_Taken INT,
    Business_Seats_Taken INT,
    FirstClass_Seats_Taken INT,
    CONSTRAINT Flt_Avail_PK
    PRIMARY KEY (Flight_ID, Segment_Number, Flight_Date)
);
0 rows inserted/updated/deleted

splice> CREATE TABLE EMP.AVAILABILITY(
    Hotel_ID INT NOT NULL,
    Booking_Date DATE NOT NULL,
    Rooms_Taken INT,
    CONSTRAINT HotelAvail_PK PRIMARY KEY (Hotel_ID, Booking_Date)
);
0 rows inserted/updated/deleted
```

See Also

- » [DROP SCHEMA statement](#)
- » [Schema Name](#)
- » [SET SCHEMA statement](#)

CREATE SEQUENCE

The `CREATE SEQUENCE` statement creates a sequence generator, which is a mechanism for generating exact numeric values, one at a time.

Syntax

```
CREATE SEQUENCE
[ SQL Identifier ]
[ sequenceElement ]*
```

The sequence name is composed of an optional *schemaName* and a *SQL Identifier*. If a *schemaName* is not provided, the current schema is the default schema. If a qualified sequence name is specified, the schema name cannot begin with SYS.

schemaName

The name of the schema to which this sequence belongs. If you do not specify a schema name, the current schema is assumed.

You cannot use a schema name that begins with the `SYS.` prefix.

SQL Identifier

The name of the sequence

sequenceElement

```
{
  AS dataType
  | START WITH startValue
  | INCREMENT BY incrementValue
  | MAXVALUE maxValue | NO MAXVALUE
  | MINVALUE minValue | NO MINVALUE
  | CYCLE | NO CYCLE
}
```

dataType

If specified, the *dataType* must be an integer type (`SMALLINT`, `INT`, or `BIGINT`). If not specified, the default data type is `INT`.

startValue

If specified, this is a signed integer representing the first value returned by the sequence object. The `START` value must be a value less than or equal to the maximum and greater than or equal to the minimum value of the sequence object.

The default start value for a new ascending sequence object is the minimum value. The default start value for a descending sequence object is the maximum value.

incrementValue

If specified, the *incrementValue* is a non-zero signed integer value that fits in a *DataType* value.

If this is not specified, the `INCREMENT` defaults to 1. If `incrementValue` is positive, the sequence numbers get larger over time; if it is negative, the sequence numbers get smaller over time.

`minValue`

If specified, `minValue` must be a signed integer that fits in a *DataType* value.

If `minValue` is not specified, or if `NO MINVALUE` is specified, then `minValue` defaults to the smallest negative number that fits in a *DataType* value.

`maxValue`

If specified, `maxValue` must be a signed integer that fits in a *DataType* value.

If `maxValue` is not specified, or if `NO MAXVALUE` is specified, then `maxValue` defaults to the largest positive number that fits in a *DataType* value.

Note that the `maxValue` must be greater than the `minValue`.

`CYCLE`

The `CYCLE` clause controls what happens when the sequence generator exhausts its range and wraps around.

If `CYCLE` is specified, the wraparound behavior is to reinitialize the sequence generator to its `START` value.

If `NO CYCLE` is specified, Splice Machine throws an exception when the generator wraps around. The default behavior is `NO CYCLE`.

To retrieve the next value from a sequence generator, use a `NEXT VALUE FOR` expression.

Usage Privileges

The owner of the schema where the sequence generator lives automatically gains the `USAGE` privilege on the sequence generator, and can grant this privilege to other users and roles. Only the database owner and the owner of the sequence generator can grant these `USAGE` privileges. The `USAGE` privilege cannot be revoked from the schema owner. See [GRANT statement](#) and [REVOKE statement](#) for more information.

Performance

To boost performance and concurrency, Splice Machine pre-allocates ranges of upcoming values for sequences. The lengths of these ranges can be configured by adjusting the value of the `derby.language.sequence.preallocator` property.

Examples

The following statement creates a sequence generator of type `INT`, with a start value of `-2147483648` (the smallest `INT` value). The value increases by 1, and the last legal value is the largest possible `INT`. If `NEXT VALUE FOR` is invoked on the generator after it reaches its maximum value, Splice Machine throws an exception.

```
splice> CREATE SEQUENCE order_id;
0 rows inserted/updated/deleted
```

This example creates a player ID sequence that starts with the integer value 100:

```
splice> CREATE SEQUENCE PlayerID_seq
    START WITH 100;
0 rows inserted/updated/deleted
```

The following statement creates a sequence of type BIGINT with a start value of 3,000,000,000. The value increases by 1, and the last legal value is the largest possible BIGINT. If `NEXT VALUE FOR` is invoked on the generator after it reaches its maximum value, Splice Machine throws an exception.

```
splice> CREATE SEQUENCE order_entry_id
    AS BIGINT
    START WITH 3000000000;
0 rows inserted/updated/deleted
```

See Also

- » [DROP SEQUENCE statement](#)
- » [GRANT statement](#)
- » [Next Value For expression](#)
- » [REVOKE statement](#)
- » [Schema Name](#)
- » [SQL Identifier](#)

CREATE SYNONYM

The CREATE SYNONYM statement allows you to create an alternate name for a table or a view.

Syntax

```
CREATE SYNONYM( tableName } );
```

synonymName

An SQLIdentifier, which can optionally include a schema name. This is the new name you want to create for the view or table.

viewName

An SQLIdentifier that identifies the view for which you are creating a synonym.

tableName

An SQLIdentifier that identifies the table for which you are creating a synonym.

Usage

NOTE: Currently, you can only use a synonym instead of the original qualified table or view name in these statements: [DELETE](#).

Here are a few other important notes about using synonyms:

- » Synonyms share the same name space as tables or views. You cannot create a synonym with the same name as a table that already exists in the same schema. Similarly, you cannot create a table or view with a name that matches a synonym already present.
- » You can create a synonym for a table or view that does not yet exist; however, you can only use the synonym if the table or view is present in your database.
- » You can create synonyms for other synonyms (nested synonyms); however, an error will occur if you attempt to create a synonym that results in a circular reference.
- » You cannot create synonyms in system schemas. Any schema that starts with SYS is a system schema.
- » You cannot define a synonym for a temporary table.

Example

```
splice> CREATE SYNONYM Hitting FOR Batting;
0 rows inserted/updated/deleted

splice> SELECT ID, Games FROM Batting WHERE ID < 11;
ID      |GAMES
-----
1       |150
2       |137
3       |100
4       |143
5       |149
6       |93
7       |133
8       |52
9       |115
10      |100

0 rows inserted/updated/deleted

splice> SELECT ID, Games FROM Hitting WHERE ID < 11;
ID      |GAMES
-----
1       |150
2       |137
3       |100
4       |143
5       |149
6       |93
7       |133
8       |52
9       |115
10      |100

0 rows inserted/updated/deleted
```

See Also

- » [DROP_SYNONYM statement](#)
- » [SHOW SYNONYMS command in our *Developer's Guide*.](#)

CREATE TABLE

A `CREATE TABLE` statement creates a table. Tables contain columns and constraints, rules to which data must conform. Table-level constraints specify a column or columns. Columns have a data type and can specify column constraints (column-level constraints).

The table owner and the database owner automatically gain the following privileges on the table and are able to grant these privileges to other users:

- » `INSERT`
- » `SELECT`
- » `TRIGGER`
- » `UPDATE`

These privileges cannot be revoked from the table and database owners.



Only database and schema owners can use the `CREATE TABLE` statement, which means that table creation privileges cannot be granted to others.

For information about constraints, see [CONSTRAINT](#) clause.

You can specify a default value for a column. A default value is the value to be inserted into a column if no other value is specified. If not explicitly specified, the default value of a column is `NULL`.

If a qualified table name is specified, the schema name cannot begin with `SYS`.

NOTE: The PIN TABLE statements are documented separately in this section.

Syntax

There are two different variants of the `CREATE TABLE` statement, depending on whether you are specifying the column definitions and constraints, or whether you are modeling the columns after the results of a query expression with the `CREATE TABLE AS` form:

```

CREATE TABLE table-Name
{
  ( {column-definition |  

    Table-level constraint}  

    [ , {column-definition} ] *  

  )  

  |  

  [ ( column-name ]* ) ]  

AS query-expression [AS <name>]  

WITH NO DATA
}

```

table-Name

The name to assign to the new table.

column-definition

A column definition.

The maximum number of columns allowed in a table is 100000.

Table-level constraint

A constraint that applies to the table.

column-name

A column definition.

AS query-expression

See the [CREATE TABLE AS](#) section below.

If this select list contains an expression, you must name the result of the expression. Refer to the final example at the bottom of this topic page.

WITH NO DATA

See the [CREATE TABLE AS](#) section below.

CREATE TABLE ... AS ...

With this alternate form of the CREATE TABLE statement, the column names and/or the column data types can be specified by providing a query. The columns in the query result are used as a model for creating the columns in the new table.

You cannot include an ORDER BY clause in the query expression you use in the CREATE TABLE AS statement.

NOTE: If the select list contains an expression, **you must name the result of the expression**. Refer to the final example at the bottom of this topic page.

If no column names are specified for the new table, then all the columns in the result of the query expression are used to create same-named columns in the new table, of the corresponding data type(s). If one or more column names are specified for the new table, then the same number of columns must be present in the result of the query expression; the data types of those columns are used for the corresponding columns of the new table.

The `WITH NO DATA` clause specifies that the data rows which result from evaluating the query expression are not used; only the names and data types of the columns in the query result are used.

There is currently a known problem using the `CREATE TABLE AS` form of the `CREATE TABLE` statement when the data to be inserted into the new table results from a `RIGHT OUTER JOIN` operation. For example, the following statement currently produces a table with all `NUL` values:

```
splice> CREATE TABLE t3 AS
    SELECT t1.a,t1.b,t2.c,t2.d
    FROM t1 RIGHT OUTER JOIN t2 ON t1.b = t2.c
    WITH DATA;
0 rows inserted/updated/deleted
```

There's a simple workaround for now: create the table without inserting the data, and then insert the data; for example:

```
splice> CREATE TABLE t3 AS
    SELECT t1.a,t1.b,t2.c,t2.d
    FROM t1 RIGHT OUTER JOIN t2 ON t1.b = t2.c
    WITH NO DATA;
0 rows inserted/updated/deleted

splice> INSERT INTO t3
    SELECT t1.a,t1.b,t2.c,t2.d
    FROM t1 RIGHT OUTER JOIN t2 ON t1.b = t2.c;
0 rows inserted/updated/deleted
```

Examples

This section presents examples of both forms of the `CREATE TABLE` statement.

CREATE TABLE

This example creates our Players table:

```
splice> CREATE TABLE Players(
    ID          SMALLINT NOT NULL PRIMARY KEY,
    Team        VARCHAR(64) NOT NULL,
    Name        VARCHAR(64) NOT NULL,
    Position    CHAR(2),
    DisplayName VARCHAR(24),
    BirthDate   DATE
);
0 rows inserted/updated/deleted
```

This example includes a table-level primary key definition that includes two columns:

```
splice> CREATE TABLE HOTELAVAILABILITY (
    Hotel_ID INT NOT NULL,
    Booking_Date DATE NOT NULL,
    Rooms_Taken INT DEFAULT 0,
    PRIMARY KEY (Hotel_ID, Booking_Date ));
0 rows inserted/updated/deleted
```

This example assigns an identity column attribute with an initial value of 5 that increments by 5, and also includes a primary key constraint:

```
splice> CREATE TABLE PEOPLE (
    Person_ID INT NOT NULL GENERATED ALWAYS AS IDENTITY (START WITH 5, INCREMENT BY 5)
        CONSTRAINT People_PK PRIMARY KEY,
    Person VARCHAR(26));
0 rows inserted/updated/deleted
```

NOTE: For more examples of CREATE TABLE statements using the various constraints, see [CONSTRAINT](#) clause

CREATE TABLE AS

This example creates a new table that uses all of the columns (and their data types) from an existing table, but does not duplicate the data:

```
splice> CREATE TABLE NewPlayers
    AS SELECT *
        FROM Players WITH NO DATA;
0 rows inserted/updated/deleted
```

This example creates a new table that includes the data and uses only some of the columns from an existing table, and assigns new names for the columns:

```
splice> CREATE TABLE MorePlayers (ID, PlayerName, Born)
    AS SELECT ID, DisplayName, Birthdate
        FROM Players WITH DATA;
94 rows inserted/updated/deleted
```

This example creates a new table using unnamed expressions in the query and shows that the data types are the same for the corresponding columns in the newly created table:

```
splice> CREATE TABLE T3 (X,Y)
    AS SELECT 2*I AS COL1, 2.0*F AS COL2
        FROM T1 WITH NO DATA;
0 rows inserted/updated/deleted
```

See Also

- » [ALTER TABLE statement](#)
- » [CREATE EXTERNAL TABLE statement](#)
- » [PIN TABLE statement](#)
- » [CONSTRAINT clause](#)
- » [DROP TABLE statement](#)
- » [UNPIN TABLE statement](#)
- » [Foreign Keys](#) in the *Developer's Guide*.
- » [Triggers](#) in the *Developer's Guide*.

CREATE TEMPORARY TABLE

The CREATE TEMPORARY TABLE statement defines a temporary table for the current connection.

This statement is similar to the `DECLARE GLOBAL TEMPORARY TABLE` statements, but uses different syntax to provide compatibility with external business intelligence tools.

For general information and notes about using temporary tables, see the [Using Temporary Tables](#) topic in our *Developer's Guide*.

Splice Machine does not currently support creating temporary tables stored as external tables.

Syntax

```
CREATE [LOCAL | GLOBAL] TEMPORARY TABLE table-Name {
    ( {column-definition | Table-level constraint}
        [ , {column-definition} ] * )
    ( column-name [ , column-name ] * )
}
[NOLOGGING | ON COMMIT PRESERVE ROWS];
```

NOTE: Splice Machine generates a warning if you attempt to specify any other modifiers other than the NOLOGGING and ON COMMIT PRESERVE ROWS modifiers shown above.

LOCAL | GLOBAL

These values are ignored by Splice Machine, and are in place simply to provide compatibility with external tools that use this syntax.

table-Name

Names the temporary table.

Table-level constraint

A constraint that is applied to this table, as described in the Constraints clause topic.

column-definition

Specifies a column definition. See column-definition for more information.

NOTE: You cannot use generated-column-spec in column-definitions for temporary tables.

column-name

A SQL Identifier that names a column in the table.

NOLOGGING

If you specify this, operations against the temporary table will not be logged; otherwise, logging will take place as usual.

ON COMMIT PRESERVE ROWS

Specifies that the data in the temporary table is to be preserved until the session terminates.

Restrictions on Temporary Tables

You can use temporary tables just like you do permanently defined database tables, with several important exceptions and restrictions that are noted in this section.

Operational Limitations

Temporary tables have the following operational limitations:

- » exist only while a user session is alive
- » are not visible to other sessions or transactions
- » cannot be altered using the `RENAME COLUMN` statements
- » do not get backed up
- » cannot be used as data providers to views
- » cannot be referenced by foreign keys in other tables
- » are not displayed by the `SHOW TABLES` command

Also note that temporary tables persist across transactions in a session and are automatically dropped when a session terminates.

Table Persistence

Here are two important notes about temporary table persistence. Temporary tables:

- » persist across transactions in a session
- » are automatically dropped when a session terminates or expires

Examples

```
splice> CREATE GLOBAL TEMPORARY TABLE FirstAndLast(
    id INT NOT NULL PRIMARY KEY,
    firstName VARCHAR(8) NOT NULL,
    lastName VARCHAR(10) NOT NULL )
    ON COMMIT PRESERVE ROWS;
0 rows inserted/updated/deleted
```

See Also

- » [DECLARE GLOBAL TEMPORARY TABLE statement](#)
- » [Using Temporary Tables](#) in the *Developer's Guide*.

CREATE TRIGGER

A `CREATE TRIGGER` statement creates a trigger, which defines a set of actions that are executed when a database event known as the triggering event occurs on a specified table. The event can be a `INSERT`, `UPDATE`, or `DELETE` statement. When a trigger fires, the set of SQL statements that constitute the action are executed.

You can define any number of triggers for a single table, including multiple triggers on the same table for the same event. To define a trigger on a table, you must be the owner of the database, the owner of the table's schema, or have `TRIGGER` privileges on the table. You cannot define a trigger for any schema whose name begins with `SYS`.

The [Database Triggers](#) topic in our *Developer's Guide* provides additional information about database triggers.

Syntax

```
CREATE TRIGGER TriggerName
  { AFTER | BEFORE }
  { INSERT | DELETE | UPDATE [ OF column-Name [, column-Name]* ] }
ON table-Name
  [ ReferencingClause ]
  [ FOR EACH { ROW | STATEMENT } ]
Triggered-SQL-statement
```

TriggerName

The name to associate with the trigger.

AFTER | BEFORE

Triggers are defined as either `Before` or `After` triggers.

`BEFORE` triggers fire before the statement's changes are applied and before any constraints have been applied. `AFTER` triggers fire after all constraints have been satisfied and after the changes have been applied to the target table.

When a database event occurs that fires a trigger, Splice Machine performs actions in this order:

- » It fires `BEFORE` triggers.
- » It performs constraint checking (primary key, unique key, foreign key, check).
- » It performs the `INSERT`, `UPDATE`, `SELECT`, or `DELETE` operations.
- » It fires `AFTER` triggers.

When multiple triggers are defined for the same database event for the same table for the same trigger time (before or after), triggers are fired in the order in which they were created.

INSERT | DELETE | SELECT | UPDATE

Defines which database event causes the trigger to fire. If you specify `UPDATE`, you can specify which column(s) cause the triggering event.

table-Name

The name of the table for which the trigger is being defined.

ReferencingClause

A means of referring to old/new data that is currently being changed by the database event that caused the trigger to fire. See the [Referencing Clause](#) section below.

FOR EACH {ROW | STATEMENT}

A FOR EACH ROW triggered action executes once for each row that the triggering statement affects.

A FOR EACH STATEMENT trigger fires once per triggering event and regardless of whether any rows are modified by the insert, update, or delete event.

Triggered-SQL-Statement

The statement that is executed when the trigger fires. The statement has the following restrictions:

- » It must not contain any dynamic (?) parameters.
- » It cannot create, alter, or drop any table.
- » It cannot add an index to or remove an index from any table.
- » It cannot add a trigger to or drop a trigger from any table.
- » It must not commit or roll back the current transaction or change the isolation level.
- » Before triggers cannot have INSERT, UPDATE, SELECT, or DELETE statements as their action.
- » Before triggers cannot call procedures that modify SQL data as their action.
- » The NEW variable of a BEFORE trigger cannot reference a generated column.

The statement can reference database objects other than the table upon which the trigger is declared. If any of these database objects is dropped, the trigger is invalidated. If the trigger cannot be successfully recompiled upon the next execution, the invocation throws an exception and the statement that caused it to fire will be rolled back.

The Referencing Clause

Many triggered-SQL-statements need to refer to data that is currently being changed by the database event that caused them to fire. The triggered-SQL-statement might need to refer to the old (pre-change or *before*) values or to the new (post-change or *after*) values. You can refer to the data that is currently being changed by the database event that caused the trigger to fire.

Note that the referencing clause can designate only one new correlation or identifier and only one old correlation or identifier.

Transition Variables in Row Triggers

Use the transition variables OLD and NEW with row triggers to refer to a single row before (OLD) or after (NEW) modification. For example:

```
REFERENCING OLD AS DELETEDROW;
```

You can then refer to this correlation name in the triggered-SQL-statement:

```
splice> DELETE FROM HotelAvailability WHERE hotel_id = DELETEDROW.hotel_id;
```

The OLD and NEW transition variables map to a *java.sql.ResultSet* with a single row.

INSERT row triggers cannot reference an OLD row.

NOTE: DELETE row triggers cannot reference a NEW row.

Trigger Recursion

The maximum trigger recursion depth is 16.

Examples

This section presents examples of creating triggers:

A statement trigger:

```
splice> CREATE TRIGGER triggerName
  AFTER UPDATE
  ON TARGET_TABLE
  FOR EACH STATEMENT
    INSERT INTO AUDIT_TABLE VALUES (CURRENT_TIMESTAMP, 'TARGET_TABLE was updated');
0 rows inserted/updated/deleted
```

A statement trigger calling a custom stored procedure:

```
splice> CREATE TRIGGER triggerName
  AFTER UPDATE
  ON TARGET_TABLE
  FOR EACH STATEMENT
    CALL my_custom_stored_procedure('arg1', 'arg2');
0 rows inserted/updated/deleted
```

A simple row trigger:

```
splice> CREATE TRIGGER triggerName
  AFTER UPDATE
  ON TARGET_TABLE
  FOR EACH ROW
    INSERT INTO AUDIT_TABLE VALUES (CURRENT_TIMESTAMP, 'TARGET_TABLE row was updated');
0 rows inserted/updated/deleted
```

A row trigger defined on a subset of columns:

```
splice> CREATE TRIGGER triggerName
    AFTER UPDATE OF col1, col2
    ON TARGET_TABLE
    FOR EACH ROW
        INSERT INTO AUDIT_TABLE VALUES (CURRENT_TIMESTAMP, 'TARGET_TABLE col1 or col2
of row was updated');
0 rows inserted/updated/deleted
```

```
splice> CREATE TRIGGER UpdateSingles
    AFTER UPDATE OF Hits, Doubles, Triples, Homeruns
    ON Batting
    FOR EACH ROW
        UPDATE Batting Set Singles=(Hits-(Doubles+Triples+Homeruns));
0 rows insert/updated/deleted
```

A row trigger defined on a subset of columns, referencing new and old values:

```
splice> CREATE TRIGGER triggerName
    AFTER UPDATE OF col1, col2
    ON T
    REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
    FOR EACH ROW
        INSERT INTO AUDIT_TABLE VALUES (CURRENT_TIMESTAMP, 'TARGET_TABLE row was updated',
        OLD_ROW.col1, NEW_ROW.col1);
0 rows insert/updated/deleted
```

A row trigger defined on a subset of columns, referencing new and old values, calling custom stored procedure:

```
splice> CREATE TRIGGER triggerName
    AFTER UPDATE OF col1, col2
    ON T
    REFERENCING OLD AS OLD_ROW NEW AS NEW_ROW
    FOR EACH ROW
        CALL my_custom_stored_procedure('arg1', 'arg2', OLD_ROW.col1, NEW_ROW.col1);
0 rows insert/updated/deleted
```

See Also

- » [Database Triggers](#) in the *Developer's Guide*
- » [DROP TRIGGER statement](#)
- » [WHERE clause](#)

CREATE VIEW

Views are virtual tables formed by a query. A view is a dictionary object that you can use until you drop it. Views are not updatable.

If a qualified view name is specified, the schema name cannot begin with SYS.

Syntax

```
CREATE VIEW view-Name
  [ ( Simple-column-Name * ) ]
  AS ORDER BY clause
  [ RESULT OFFSET clause ]
  [ FETCH FIRST clause ]
```

A view definition can contain an optional view column list to explicitly name the columns in the view. If there is no column list, the view inherits the column names from the underlying query. All columns in a view must be uniquely named.

view-Name

The name to assign to the view.

*Simple-column-Name**

An optional list of names to be used for columns of the view. If not given, the column names are deduced from the query.

The maximum number of columns in a view is 5000.

AS Query [ORDER BY clause]

A SELECT or VALUES command that provides the columns and rows of the view.

result offset and fetch first clauses

The **FETCH FIRST** clause, which can be combined with the **RESULT OFFSET** clause, limits the number of rows added to the view.

Examples

This example creates a view that shows the age of each player in our database:

```
splice> CREATE VIEW PlayerAges (Player, Team, Age)
    AS SELECT DisplayName, Team,
        INT( (Now - Birthdate) / 365.25) AS Age
    FROM Players;
0 rows inserted/updated/deleted
```

```
splice> SELECT * FROM PlayerAges WHERE Age > 30 ORDER BY Team, Age DESC;
```

PLAYER	TEAM	AGE
Robert Cohen	Cards	40
Jason Larrimore	Cards	37
David Janssen	Cards	36
Mitch Hassleman	Cards	35
Mitch Brandon	Cards	35
Tam Croonster	Cards	34
Alex Wister	Cards	34
Yuri Milleton	Cards	33
Jonathan Pearlman	Cards	33
Michael Rastono	Cards	32
Barry Morse	Cards	32
Carl Vanamos	Cards	32
Jan Bromley	Cards	31
Thomas Hillman	Giants	40
Mark Briste	Giants	38
Randy Varner	Giants	38
Jason Lilliput	Giants	38
Jalen Ardson	Giants	36
Sam Castleman	Giants	35
Alex Paramour	Giants	34
Jack Peepers	Giants	34
Norman Aikman	Giants	33
Craig McGawn	Giants	33
Kameron Fannais	Giants	33
Jason Martell	Giants	33
Harry Pennello	Giants	32
Jason Minman	Giants	32
Trevor Imhof	Giants	32
Steve Raster	Giants	32
Greg Brown	Giants	31
Alex Darba	Giants	31
Joseph Arkman	Giants	31
Tam Lassiter	Giants	31
Martin Cassman	Giants	31
Yuri Piamam	Giants	31

35 rows selected

Statement Dependency System

View definitions are dependent on the tables and views referenced within the view definition. DML (data manipulation language) statements that contain view references depend on those views, as well as the objects in the view definitions that the views are dependent on. Statements that reference the view depend on indexes the view uses; which index a view uses can change from statement to statement based on how the query is optimized. For example, given:

```
splice> CREATE TABLE T1 (C1 DOUBLE PRECISION);
0 rows inserted/updated/deleted

splice>CREATE FUNCTION SIN (DATA DOUBLE)
    RETURNS DOUBLE
    EXTERNAL NAME 'java.lang.Math.sin'
    LANGUAGE JAVA PARAMETER STYLE JAVA;
0 rows inserted/updated/deleted

splice> CREATE VIEW V1 (C1) AS SELECT SIN(C1) FROM T1;
0 rows inserted/updated/deleted
```

The following SELECT:

```
SELECT * FROM V1;
```

Is dependent on view *V1*, table *T1*, and external scalar function *SIN*.

See Also

- » [DROP VIEW statement](#)
- » [ORDER BY clause](#)

DECLARE GLOBAL TEMPORARY TABLE

The DECLARE GLOBAL TEMPORARY TABLE statement defines a temporary table for the current connection.

This statement is similar to the [CREATE GLOBAL TEMPORARY TABLE](#) and CREATE LOCAL TEMPORARY TABLE statements, but uses different syntax to provide compatibility with external business intelligence tools.

For general information and notes about using temporary tables, see the [Using Temporary Tables](#) topic in our *Developer's Guide*.

Syntax

```
DECLARE GLOBAL TEMPORARY TABLE table-Name
  { column-definition[ , column-definition] * }
    [ON COMMIT PRESERVE ROWS ]
    [NOT LOGGED]
```

NOTE: Splice Machine generates a warning if you attempt to specify any other modifiers other than the NOT LOGGED and ON COMMIT PRESERVE ROWS modifiers shown above.

table-Name

Names the temporary table.

column-definition

Specifies a column definition. See column-definition for more information.

NOTE: You cannot use generated-column-spec in column-definitions for temporary tables.

ON COMMIT PRESERVE ROWS

Specifies that the data in the temporary table is to be preserved until the session terminates.

NOT LOGGED

If you specify this, operations against the temporary table will not be logged; otherwise, logging will take place as usual.

Restrictions on Temporary Tables

You can use temporary tables just like you do permanently defined database tables, with several important exceptions and restrictions that are noted in this section.

Operational Limitations

Temporary tables have the following operational limitations:

- » exist only while a user session is alive
- » are not visible to other sessions or transactions
- » cannot be altered using the `RENAME COLUMN` statements
- » do not get backed up
- » cannot be used as data providers to views
- » cannot be referenced by foreign keys in other tables
- » are not displayed by the `SHOW TABLES` command

Also note that temporary tables persist across transactions in a session and are automatically dropped when a session terminates.

Table Persistence

Here are two important notes about temporary table persistence. Temporary tables:

- » persist across transactions in a session
- » are automatically dropped when a session terminates or expires

Examples

```
splice> DECLARE GLOBAL TEMPORARY TABLE FirstAndLast(
    id INT NOT NULL PRIMARY KEY,
    firstName VARCHAR(8) NOT NULL,
    lastName VARCHAR(10) NOT NULL )
ON COMMIT PRESERVE ROWS
NOT LOGGED;
0 rows inserted/updated/deleted
```

See Also

- » [CREATE TEMPORARY TABLE](#) statement
- » [Using Temporary Tables](#) in the *Developer's Guide*.

DELETE

The DELETE statement deletes records from a table.

Our [Bulk HFile Delete](#) feature can be used to optimize deletion of large amounts of data.

Syntax

```
{
    DELETE FROM correlation-Name]
    [WHERE clause]
}
```

table-Name

The name of the table from which you want to delete records.

correlation-Name

The optional alias (alternate name) for the table.

WHERE clause

The clause that specifies which record(s) to select for deletion.

Usage

The DELETE statement removes all rows identified by the table name and *WHERE* clause.

Examples

```
splice> DELETE FROM Players WHERE Year(Birthdate) > 1990;
8 rows inserted/updated/deleted
```

Using our Bulk HFile Delete Feature

Our Bulk Delete feature leverages HFile bulk deletion to significantly speed things up when you are deleting a lot of data; it does so by generating HFiles for the deletion and then bypasses the Splice Machine write pipeline and HBase write path when deleting the data.

You simply add a [splice-properties](#) hint that specifies where to generate the HFiles. If you're specifying an S3 bucket on AWS, please review our Configuring an S3 Bucket for Splice Machine Access tutorial before proceeding.

```
splice> DELETE FROM my_table --splice-properties bulkDeleteDirectory='/bulkFilesPat  
h'  
;
```

NOTE: We recommend performing a major compaction on your database after deleting a large amount of data; you should also be aware of our new [SYSCS_UTIL.SET_PURGE_DELETED_ROWS](#) system procedure, which you can call before a compaction to specify that you want the data physically (not just logically) deleted during compaction.

Statement Dependency System

A searched delete statement depends on the table being updated, all of its conglomerates (units of storage such as heaps or indexes), and any other table named in the WHERE clause. A [DROP INDEX](#) statement for the target table of a prepared searched delete statement invalidates the prepared searched delete statement.

A [CREATE INDEX](#) or [DROP INDEX](#) statement for the target table of a prepared positioned delete invalidates the prepared positioned delete statement.

See Also

- » [CREATE INDEX](#) statement
- » [DROP INDEX](#) statement
- » [SELECT](#) statement
- » [WHERE](#) clause

Interaction with the Dependency System

Splice Machine internally tracks the dependencies of prepared statements, which are SQL statements that are precompiled before being executed. Typically they are prepared (precompiled) once and executed multiple times.

Prepared statements depend on the dictionary objects and statements they reference. (Dictionary objects include tables, columns, constraints, indexes, and views, and triggers. Removing or modifying the dictionary objects or statements on which they depend invalidates them internally, which means that Splice Machine will automatically try to recompile the statement when you execute it. If the statement fails to recompile, the execution request fails. However, if you take some action to restore the broken dependency (such as restoring the missing table), you can execute the same prepared statement, because Splice Machine will recompile it automatically at the next execute request.

Statements depend on one another—an UPDATE WHERE CURRENT statement depends on the statement it references. Removing the statement on which it depends invalidates the UPDATE WHERE CURRENT statement.

In addition, prepared statements prevent execution of certain DDL statements if there are open results sets on them.

Manual pages for each statement detail what actions would invalidate that statement, if prepared. Here is an example using The Splice Machine command line interface:

```

splice> CREATE TABLE mytable (mycol INT);
      0 rows inserted/updated/deleted
splice> INSERT INTO mytable VALUES (1), (2), (3);
      3 rows inserted/updated/deleted -- this example uses the
ij command prepare, which prepares a statement
splice> prepare p1 AS 'INSERT INTO MyTable VALUES (4)';
      -- p1 depends on mytable;
splice> execute p1;
      1 row inserted/updated/deleted
      -- Splice Machine executes it without recompiling
splice> CREATE INDEX i1 ON mytable(mycol);
      0 rows inserted/updated/deleted
      -- p1 is temporarily invalidated because of new index
splice> execute p1;
      1 row inserted/updated/deleted
      -- Splice Machine automatically recompiles and executes p1
splice> DROP TABLE mytable;
      0 rows inserted/updated/deleted
      -- Splice Machine permits you to drop table
      -- because result set of p1 is closed
      -- however, the statement p1 is temporarily invalidated
splice> CREATE TABLE mytable (mycol INT);
      0 rows inserted/updated/deleted
splice> INSERT INTO mytable VALUES (1), (2), (3);
      3 rows inserted/updated/deleted
splice> execute p1;
      1 row inserted/updated/deleted
      -- p1 is invalid, so Splice Machine tries to recompile it
      -- before executing.
      -- It is successful and executes.
splice> DROP TABLE mytable;
      0 rows inserted/updated/deleted
      -- statement p1 is now invalid
      -- and this time the attempt to recompile it
      -- upon execution will fail
splice> execute p1;
      ERROR 42X05: Table/View 'MYTABLE' does not exist.

```

See Also

- » [CREATE INDEX statement](#)
- » [CREATE TABLE statement](#)
- » [DROP TABLE statement](#)
- » [INSERT statement](#)
- » [Using the `splice>` prompt](#)

DROP FUNCTION

The `DROP FUNCTION` statement drops a function from your database. Functions are added to the database with the [CREATE FUNCTION](#) statement.

Syntax

```
DROP FUNCTION function-name
```

function-name

The name of the function that you want to drop from your database.

Usage

Use this statement to drop a function from your database. It is valid only if there is exactly one function instance with the *function-name* in the schema. The specified function can have any number of parameters defined for it.

An error will occur in any of the following circumstances:

- » If no function with the indicated name exists in the named or implied schema (the error is SQLSTATE 42704)
- » If there is more than one specific instance of the function in the named or implied schema
- » If you try to drop a user-defined function that is invoked in the *generation-clause* of a generated column
- » If you try to drop a user-defined function that is invoked in a view

Example

```
splice> DROP FUNCTION TO_DEGREES;  
0 rows inserted/updated/deleted
```

See Also

- » [CREATE FUNCTION](#) statement

DROP INDEX

The `DROP INDEX` statement removes the specified index.

Syntax

```
DROP INDEX index-Name
```

index-Name

The name of the index that you want to drop from your database.

Examples

```
splice> DROP INDEX myIdx;  
0 rows inserted/updated/deleted
```

See Also

- » [CREATE_INDEX statement](#)
- » [DELETE statement](#)
- » [INSERT statement](#)
- » [SELECT statement](#)
- » [UPDATE statement](#)

DROP PROCEDURE

The `DROP PROCEDURE` statement drops a procedure from your database. Procedures are added to the database with the [CREATE PROCEDURE](#) statement.

Syntax

```
DROP PROCEDURE procedure-name
```

procedure-Name

The name of the procedure that you want to drop from your database.

Usage

Use this statement to drop a statement from your database. It is valid only if there is exactly one procedure instance with the *procedure-name* in the schema. The specified procedure can have any number of parameters defined for it.

An error will occur in any of the following circumstances:

- » If no procedure with the indicated name exists in the named or implied schema (the error is SQLSTATE 42704)
- » If there is more than one specific instance of the procedure in the named or implied schema
- » If you try to drop a user-defined procedure that is invoked in the *generation-clause* of a generated column
- » If you try to drop a user-defined procedure that is invoked in a view

Example

```
splice> DROP PROCEDURE SALES.TOTAL_REVENUE;
0 rows inserted/updated/deleted
```

See Also

- » Argument matching
- » [CREATE_PROCEDURE](#) statement
- » [CURRENT_USER](#) function
- » Data Types
- » Schema Name
- » SQL Identifier

» [SESSION_USER function](#)

» [USER function](#)

DROP ROLE

The `DROP ROLE` statement allows you to drop a role from your database.

Syntax

```
DROP ROLE roleName
```

roleName

The name of the role that you want to drop from your database.

Usage

Dropping a role has the effect of removing the role from the database dictionary. This means that no session user can henceforth set that role (see `CURRENT_ROLE` function) will now have a `NULL CURRENT_ROLE`.

Dropping a role also has the effect of revoking that role from any user and role it has been granted to. See the `REVOKE` statement for information on how revoking a role may impact any dependent objects.

Example

```
splice> DROP ROLE statsEditor_role;  
0 rows inserted/updated/deleted
```

See Also

- » [CREATE_ROLE statement](#)
- » [GRANT statement](#)
- » [REVOKE statement](#)
- » [SET ROLE statement](#)
- » [SYSROLES system table](#)

DROP SCHEMA

The `DROP SCHEMA` statement drops a schema. The target schema must be empty for the drop to succeed.

Neither the `SPLICE` schema (the default user schema) nor the `SYS` schema can be dropped.

Syntax

```
DROP SCHEMA schemaName RESTRICT
```

schema

The name of the schema that you want to drop from your database.

RESTRICT

This is **required**. It enforces the rule that the schema cannot be deleted from the database if there are any objects defined in the schema.

Example

```
splice> DROP SCHEMA Baseball_Stats RESTRICT;
0 rows inserted/updated/deleted
```

See Also

- » [CREATE SCHEMA statement](#)
- » [Schema Name](#)
- » [SET SCHEMA statement](#)

DROP SEQUENCE

The `DROP SEQUENCE` statement removes a sequence generator that was created using a `CREATE SEQUENCE` statement.

Syntax

```
DROP SEQUENCE [ schemaName "." ] SQL Identifier RESTRICT
```

schemaName

The name of the schema to which this sequence belongs. If you do not specify a schema name, the current schema is assumed.

You cannot use a schema name that begins with the `SYS.` prefix.

SQL Identifier

The name of the sequence.

RESTRICT

This is **required**. It specifies that if a trigger or view references the sequence generator, Splice Machine will throw an exception.

Usage

Dropping a sequence generator implicitly drops all `USAGE` privileges that reference it.

Example

```
splice> DROP SEQUENCE PLAYERID_SEQ RESTRICT;
0 rows inserted/updated/deleted
```

See Also

- » [CREATE SEQUENCE statement](#)
- » [Schema Name](#)

DROP SYNONYM

The `DROP SYNONYM` statement drops a synonym that was previously defined for a table or view.

Syntax

```
DROP SYNONYM synonymName
```

synonymName

The name of the synonym that you want to drop from your database.

Example

```
splice> DROP SYNONYM Hitting;  
0 rows inserted/updated/deleted
```

See Also

- » [CREATE_SYNONYM statement](#)
- » [SHOW SYNONYMS command in our *Developer's Guide*.](#)

DROP TABLE

The `DROP TABLE` statement removes the specified table.

Syntax

```
DROP TABLE [ IF EXISTS ] table-Name
```

table-Name

The name of the schema that you want to drop from your database.

Statement dependency system

Indexes and constraints , constraints (primary, unique, check and references from the table being dropped) and triggers on the table are silently dropped.

Dropping a table invalidates statements that depend on the table. (Invalidating a statement causes it to be recompiled upon the next execution. See Interaction with the dependency system.)

Example

```
splice> DROP TABLE Salaries;
0 rows inserted/updated/deleted
```

See Also

- » [ALTER TABLE](#) statement
- » [CREATE TABLE](#) statement
- » [CONSTRAINT](#) clause

DROP TRIGGER

The `DROP TRIGGER` statement removes the specified trigger.

Syntax

```
DROP TRIGGER TriggerName
```

TriggerName

The name of the trigger that you want to drop from your database.

Example

```
splice> DROP TRIGGER UpdateSingles;  
0 rows inserted/updated/deleted
```

Statement dependency system

When a table is dropped, all triggers on that table are automatically dropped; this means that do not have to drop a table's triggers before dropping the table.

See Also

- » [Database Triggers](#) in the *Developer's Guide*
- » [CREATE TRIGGER](#) statement

DROP VIEW

The `DROP VIEW` statement drops the specified view.

Syntax

```
DROP VIEW view-Name
```

view-Name

The name of the view that you want to drop from your database.

Example

```
splice> DROP VIEW PlayerAges;  
0 rows inserted/updated/deleted
```

Statement dependency system

Any statements referencing the view are invalidated on a `DROP VIEW` statement.

See Also

- » [CREATE VIEW statement](#)
- » [ORDER BY clause](#)

generated-column-spec

A generated column is one whose value is defined by an expression, typically involving values from other columns in the same table. The value of a generated column is automatically updated whenever there's a change in the value of any column upon which the expression depends.

```
[ GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY
[ ( START WITH IntegerConstant
[ ,INCREMENT BY IntegerConstant] ) ] ] ]
```

{ALWAYS | BY DEFAULT} AS IDENTITY

A table can have at most one identity column. See the [Identity Column Attributes](#) section below for more information about identity columns. Splice Machine supports two kinds of identity columns:

GENERATED ALWAYS

An identity column that is `GENERATED ALWAYS` will increment the default value on every insertion and will store the incremented value into the column. Unlike other defaults, you cannot insert a value directly into or update an identity column that is `GENERATED ALWAYS`. Instead, either specify the `DEFAULT` keyword when inserting into the identity column, or leave the identity column out of the insertion column list altogether. For example:

```
create table greetings
  (i int generated always as identity, ch char(50));
insert into greetings values (DEFAULT, 'hello');
insert into greetings(ch) values ('bonjour');
```

Automatically generated values in a `GENERATED ALWAYS` identity column are unique. Creating an identity column does not create an index on the column.

GENERATED BY DEFAULT

An identity column that is `GENERATED BY DEFAULT` will only increment and use the default value on insertions when no explicit value is given. Unlike `GENERATED ALWAYS` columns, you can specify a particular value in an insertion statement to be used instead of the generated default value.

To use the generated default, either specify the `DEFAULT` keyword when inserting into the identity column, or just leave the identity column out of the insertion column list. To specify a value, include it in the insertion statement. For example:

```
create table greetings
  (i int generated by default as identity, ch char(50));
    -- specify value "1":
insert into greetings values (1, 'hi');
    -- use generated default
insert into greetings values (DEFAULT, 'salut');
    -- use generated default
insert into greetings(ch) values ('bonjour');
```

Note that unlike a `GENERATED ALWAYS` column, a `GENERATED BY DEFAULT` column does not guarantee uniqueness. Thus, in the above example, the `hi` and `salut` rows will both have an identity value of “1”, because the generated column starts at 1 and the user-specified value was also 1. You can prevent duplication by specifying a `START WITH` value, and using a primary key or unique constraint on the identity column.

START WITH IntegerConstant

The first identity value that Splice Machine should assign.

INCREMENT BY IntegerConstant

The amount by which to increment the identity value each time one is assigned.

Identity Column Attributes

A table can have at most one identity column.

For `SMALLINT`, `INT`, and `BIGINT` columns with identity attributes, Splice Machine automatically assigns increasing integer values to the column. Identity column attributes behave like other defaults in that when an insert statement does not specify a value for the column, Splice Machine automatically provides the value. However, the value is not a constant; Splice Machine automatically increments the default value at insertion time.

The `IDENTITY` keyword can only be specified if the data type associated with the column is one of the following exact integer types.

- » `SMALLINT`
- » `INT`
- » `BIGINT`

By default, the initial value of an identity column is 1, and the amount of the increment is 1. You can specify any positive integer value for both the initial value and the interval amount when you define the column with the key words `START WITH` and `INCREMENT BY`. Splice Machine increments the value with each insert. A value of 0 raises a statement exception.

The maximum and minimum values allowed in identity columns are determined by the data type of the column. Attempting to insert a value outside the range of values supported by the data type raises an exception. The following table shows the supported ranges.

Data Type	Maximum Value	Minimum Value
<code>SMALLINT</code>	<code>32767 (java.lang.Short.MAX_VALUE)</code>	<code>-32768 (java.lang.Short.MIN_VALUE)</code>
<code>INT</code>	<code>2147483647 (java.lang.Integer.MAX_VALUE)</code>	<code>-2147483648 (java.lang.Integer.MIN_VALUE)</code>
<code>BIGINT</code>	<code>9223372036854775807 (java.lang.Long.MAX_VALUE)</code>	<code>-9223372036854775808 (java.lang.Long.MIN_VALUE)</code>

Automatically generated values in an identity column are unique. Use a primary key or unique constraint on a column to guarantee uniqueness. Creating an identity column *does not* create an index on the column.

NOTE: Specify the schema, table, and column name using the same case as those names are stored in the system tables—that is, all upper case unless you used delimited identifiers when creating those database objects.

Using Generated Columns

Splice Machine keeps track of the last increment value for a column in a cache. It also stores the value of what the next increment value will be for the column on disk in the AUTOINCREMENTVALUE column of the `SYS.SYSCOLUMNS` system table. Rolling back a transaction does not undo this value, and thus rolled-back transactions can leave “gaps” in the values automatically inserted into an identity column. Splice Machine behaves this way to avoid locking a row in `SYS.SYSCOLUMNS` for the duration of a transaction and keeping concurrency high.

When an insert happens within a triggered-SQL-statement, the value inserted by the triggered-SQL-statement into the identity column is available from `ConnectionInfo` only within the trigger code. The trigger code is also able to see the value inserted by the statement that caused the trigger to fire. However, the statement that caused the trigger to fire is not able to see the value inserted by the triggered-SQL-statement into the identity column. Likewise, triggers can be nested (or recursive).

An SQL statement can cause trigger T1 to fire. T1 in turn executes an SQL statement that causes trigger T2 to fire. If both T1 and T2 insert rows into a table that cause Splice Machine to insert into an identity column, trigger T1 cannot see the value caused by T2’s insert, but T2 can see the value caused by T1’s insert. Each nesting level can see increment values generated by itself and previous nesting levels, all the way to the top-level SQL statement that initiated the recursive triggers. You can only have 16 levels of trigger recursion.

Examples

```

create table greetings
  (i int generated by default
   as identity (START WITH 2, INCREMENT BY 1),
   ch char(50));
-- specify value "1":
insert into greetings values (1, 'hi');
-- use generated default
insert into greetings values (DEFAULT, 'salut');
-- use generated default
insert into greetings(ch) values ('bonjour');
drop table if exists words;

splice> CREATE TABLE WORDS(WORD VARCHAR(20), UWORD GENERATED ALWAYS AS (UPPER(WORD)));
0 rows inserted/updated/deleted
splice> CREATE INDEX IDX_UWORD ON WORDS(UWORD);
0 rows inserted/updated/deleted
splice> INSERT INTO WORDS(WORD) VALUES 'chocolate', 'Coca-Cola', 'hamburger', 'carrot';
4 rows inserted/updated/deleted
splice> select * from words;
WORD          |UWORD
-----
chocolate      |CHOCOLATE
Coca-Cola      |COCA-COLA
hamburger      |HAMBURGER
carrot         |CARROT

4 rows selected
splice> select upper(word) from words;
1
-----
CHOCOLATE
COCA-COLA
HAMBURGER
CARROT

4 rows selected
splice> drop table if exists t;
0 rows inserted/updated/deleted
WARNING 42Y55: 'DROP TABLE' cannot be performed on 'T' because it does not exist.
splice> CREATE TABLE T(COL1 INT, COL2 INT, COL3 GENERATED ALWAYS AS (COL1+COL2));
0 rows inserted/updated/deleted
splice> INSERT INTO T (COL1, COL2) VALUES (1,2), (3,4), (5,6);
3 rows inserted/updated/deleted
splice> select * from t;
COL1          |COL2          |COL3
-----
1              |2              |3
3              |4              |7
5              |6              |11
-----
```

```
3 rows selected
splice> UPDATE T SET COL2 = 100 WHERE COL1 = 1;
1 row inserted/updated/deleted
splice> select * from t;
COL1      | COL2      | COL3
-----
1          | 100       | 101
3          | 4          | 7
5          | 6          | 11
3 rows selected
```

generation-clause

Syntax

```
GENERATED ALWAYS AS ( value-expression )
```

value-expression

An *Expression* that resolves to a single value, with some limitations:

- » The *generation-clause* may reference other non-generated columns in the table, but it must not reference any generated column. The *generation-clause* must not reference a column in another table.
- » The *generation-clause* must not include subqueries.
- » The *generation-clause* may invoke user-coded functions, if the functions meet the requirements in the [User Function Restrictions](#) section below.

User Function Restrictions

The *generation-clause* may invoke user-coded functions, if the functions meet the following requirements:

- » The functions must not read or write SQL data.
- » The functions must have been declared DETERMINISTIC.
- » The functions must not invoke any of the following possibly non-deterministic system functions:
 - » SESSION_USER

Example

```
CREATE TABLE employee(
    employeeID          int,
    name                varchar( 50 ),
    caseInsensitiveName GENERATED ALWAYS AS( UPPER( name ) )
);

CREATE INDEX caseInsensitiveEmployeeName
    ON employee( caseInsensitiveName );
```

GRANT

Use the GRANT statement to give privileges to a specific user or role, or to all users, to perform actions on database objects. You can also use the GRANT statement to grant a role to a user, to PUBLIC, or to another role.

The syntax that you use for the GRANT statement depends on whether you are granting privileges to a schema object or granting a role.



Only database and schema owners can use the `CREATE TABLE` statement, which means that table creation privileges cannot be granted to others, even with `GRANT ALL PRIVILEGES`.

Syntax for Schemas

```
GRANT ALL PRIVILEGES | schema-privilege {, schema-privilege }*
  ON [SCHEMA] schema-Name
  TO grantees
```

schema-privilege

```
DELETE
| INSERT
| REFERENCES [( column-identifier {, column-identifier}* )]
| SELECT [( column-identifier {, column-identifier}* )]
| TRIGGER
| UPDATE [( column-identifier {, column-identifier}* )]
```

See the [Privilege Types](#) section below for more information.



Column-level privileges are available only with a Splice Machine Enterprise license.

You cannot grant or revoke privileges at the `column-identifier` level with the Community version of Splice Machine.

To obtain a license for the Splice Machine Enterprise Edition, please [Contact Splice Machine Sales](#) today.

schema-Name

The name of the schema to which you are granting access.

grantees

The user(s) or role(s) to whom you are granting access. See the [About Grantees](#) section below for more information.

NOTES:

- » When you drop a schema from your database, all privileges associated with the schema are removed.
- » Table-level privileges override schema-level privileges.

Syntax for Tables

```
GRANT ALL PRIVILEGES | table-privilege {, table-privilege}* ON [TABLE] { tableName
e }
TO grantees
```

table-privilege

```
DELETE
| INSERT
| REFERENCES [( column-identifier {, column-identifier}* )]
| SELECT [( column-identifier {, column-identifier}* )]
| TRIGGER
| UPDATE [( column-identifier {, column-identifier}* )]
```

See the [Privilege Types](#) section below for more information.



Column-level privileges are available only with a Splice Machine Enterprise license.

You cannot grant or revoke privileges at the `column-identifier` level with the Community version of Splice Machine.

To obtain a license for the Splice Machine Enterprise Edition, please [Contact Splice Machine Sales](#) today.

table-Name

The name of the table to which you are granting access.

view-Name

The name of the view to which you are granting access.

schema-Name

The name of the schema to which you are granting access.

grantees

The user(s) or role(s) to whom you are granting access. See the [About Grantees](#) section below for more information.

NOTES:

- » When you drop a table from your database, all privileges associated with the table are removed.

- » Table-level privileges override schema-level privileges.

Syntax for Routines

```
GRANT EXECUTE
  ON { FUNCTION | PROCEDURE } {function-name | procedure-name}
  TO grantees
```

function-name | procedure-name

The name of the function or procedure to which you are granting access.

grantees

The user(s) or role(s) to whom you are granting access. See the [About Grantees](#) section below for more information.

Syntax for User-defined Types

```
GRANT USAGE
  SQL Identifier
  TO grantees
```

[schema-name.] SQL Identifier

The type name is composed of an optional *schemaName* and a *SQL Identifier*. If a *schemaName* is not provided, the current schema is the default schema. If a qualified UDT name is specified, the schema name cannot begin with *SYS*.

grantees

The user(s) or role(s) to whom you are granting access. See the [About Grantees](#) section below for more information.

Syntax for Roles

```
GRANT roleName
  { roleName }*
  TO grantees
```

roleName

The name to the role(s) to which you are granting access.

grantees

The user(s) or role(s) to whom you are granting access. See the [About Grantees](#) section below for more information.

Before you can grant a role to a user or to another role, you must create the role using the [CREATE ROLE](#) statement. Only the database owner can grant a role.

A role A *contains* another role B if role B is granted to role A, or is contained in a role C granted to role A. Privileges granted to a contained role are inherited by the containing roles. So the set of privileges identified by role A is the union of the privileges granted to role A and the privileges granted to any contained roles of role A.

About Grantees

A grantee can be one or more specific users, one or more specific roles, or all users (`PUBLIC`). Either the object owner or the database owner can grant privileges to a user or to a role. Only the database owner can grant a role to a user or to another role.

Here's the syntax:

```
{      roleName | PUBLIC }
[, { roleName | PUBLIC } ] *
```

AuthorizationIdentifier

An expression.

roleName

The name of the role.

Either the object owner or the database owner can grant privileges to a user or to a role. Only the database owner can grant a role to a user or to another role.

`PUBLIC`

Use the keyword `PUBLIC` to specify all users.

When `PUBLIC` is specified, the privileges or roles affect all current and future users.

The privileges granted to `PUBLIC` and to individual users or roles are independent privileges. For example, a `SELECT` privilege on table `t` is granted to both `PUBLIC` and to the authorization ID `harry`. If the `SELECT` privilege is later revoked from the authorization ID `harry`, Harry will still be able to access the table `t` through the `PUBLIC` privilege.

Privilege Types

Privilege Type	Usage
ALL PRIVILEGES	To grant all of the privileges to the user or role for the specified table. You can also grant one or more table privileges by specifying a privilege-list.  Only database and schema owners can use the CREATE TABLE statement, which means that table creation privileges cannot be granted to others, even with GRANT ALL PRIVILEGES.
DELETE	To grant permission to delete rows from the specified table.
INSERT	To grant permission to insert rows into the specified table.
REFERENCES	To grant permission to create a foreign key reference to the specified table. If a column list is specified with the REFERENCES privilege, the permission is valid on only the foreign key reference to the specified columns.
SELECT	To grant permission to perform SelectExpressions on a table or view. If a column list is specified with the SELECT privilege, the permission is valid on only those columns. If no column list is specified, then the privilege is valid on all of the columns in the table. For queries that do not select a specific column from the tables involved in a SELECT statement or <i>SelectExpression</i> (for example, queries that use COUNT(*)), the user must have at least one column-level SELECT privilege or table-level SELECT privilege.
TRIGGER	To grant permission to create a trigger on the specified table.
UPDATE	To grant permission to use the WHERE clause, you must have the SELECT privilege on the columns in the row that you want to update.

Usage Notes

The following types of privileges can be granted:

- » Delete data from a specific table.
- » Insert data into a specific table.
- » Create a foreign key reference to the named table or to a subset of columns from a table.
- » Select data from a table, view, or a subset of columns in a table.

- » Create a trigger on a table.
- » Update data in a table or in a subset of columns in a table.
- » Run a specified function or procedure.
- » Use a user-defined type.

Before you issue a GRANT statement, check that the `derby.database.sqlAuthorization` property is set to `true`. The `derby.database.sqlAuthorization` property enables the SQL Authorization mode.

You can grant privileges on an object if you are the owner of the object or the database owner. See documentation for the [CREATE](#) statements for more information.

Examples

Granting Privileges to Users

To grant the SELECT privilege on the schema SpliceBBall to the authorization IDs Bill and Joan, use the following syntax:

```
splice> GRANT SELECT ON SCHEMA SpliceBBall TO Bill, Joan;
0 rows inserted/updated/deleted
```

To grant the SELECT privilege on table Salaries to the authorization IDs Bill and Joan, use the following syntax:

```
splice> GRANT SELECT ON TABLE Salaries TO Bill, Joan;
0 rows inserted/updated/deleted
```

To grant the UPDATE and TRIGGER privileges on table Salaries to the authorization IDs Joe and Anita, use the following syntax:

```
splice> GRANT UPDATE, TRIGGER ON TABLE Salaries TO Joe, Anita;
0 rows inserted/updated/deleted
```

To grant the SELECT privilege on table Hitting in the Baseball_stats schema to all users, use the following syntax:

```
splice> GRANT SELECT ON TABLE Baseball_Stats.Hitting to PUBLIC;
0 rows inserted/updated/deleted
```

To grant the EXECUTE privilege on procedure ComputeValue to the authorization ID george, use the following syntax:

```
splice> GRANT EXECUTE ON PROCEDURE ComputeValue TO george;
0 rows inserted/updated/deleted
```

Granting Roles to Users

To grant the role ` purchases_reader_role to the authorization IDs george and maria`, use the following syntax:

```
splice> GRANT purchases_reader_role TO george,maria;
0 rows inserted/updated/deleted
```

Granting Privileges to Roles

To grant the SELECT privilege on schema SpliceBBall to the role purchases_reader_role, use the following syntax:

```
splice> GRANT SELECT ON SCHEMA SpliceBBall TO purchases_reader_role;
0 rows inserted/updated/deleted
```

To grant the SELECT privilege on table t to the role purchases_reader_role, use the following syntax:

```
splice> GRANT SELECT ON TABLE t TO purchases_reader_role;
0 rows inserted/updated/deleted
```

See Also

- » [CREATE ROLE statement](#)
- » [CREATE TRIGGER statement](#)
- » [DROP_ROLE statement](#)
- » [REVOKE statement](#)
- » [RoleName](#)
- » [SET ROLE statement](#)
- » [SELECT expression](#)
- » [SELECT statement](#)
- » [SYSROLES system table](#)
- » [UPDATE statement](#)
- » [WHERE clause](#)

INSERT

An INSERT statement creates rows or columns and stores them in the named table. The number of values assigned in an INSERT statement must be the same as the number of specified or implied columns.

Whenever you insert into a table which has generated columns, Splice Machine calculates the values of those columns.

Syntax

```
INSERT INTO table-Name
  [ (Simple-column-Name)* ) ]
  Query [ ORDER BY clause ]
  [ result offset clause ]
  [ fetch first clause ];
```

table-Name

The table into which you are inserting data.

*Simple-column-Name**

An optional list of names of the columns to populate with data.

Query [ORDER BY *clause*]

A SELECT or VALUES command that provides the columns and rows of data to insert. The query can also be a UNION expression.

See the [Using the ORDER BY Clause](#) section below for information about using the ORDER BY clause.

Single-row and multiple-row VALUES expressions can include the keyword DEFAULT. Specifying DEFAULT for a column inserts the column's default value into the column. Another way to insert the default value into the column is to omit the column from the column list and only insert values into other columns in the table. For more information, see [VALUES](#) expression

result offset and fetch first clauses

The `fetch first` clause, which can be combined with the `result offset` clause, limits the number of rows added to the table.

Using the ORDER BY Clause

When you want insertion to happen with a specific ordering (for example, in conjunction with auto-generated keys), it can be useful to specify an ORDER BY clause on the result set to be inserted.

If the Query is a VALUES expression, it cannot contain or be followed by an ORDER BY, result offset, or fetch first clause. However, if the VALUES expression does not contain the DEFAULT keyword, the VALUES clause can be put in a subquery and ordered, as in the following statement:

```
INSERT INTO t SELECT * FROM (VALUES 'a','c','b') t ORDER BY 1;
```

For more information about queries, see [Query](#).

Examples

These examples insert records with literal values:

```
splice> INSERT INTO Players
    VALUES( 99, 'Giants', 'Joe Bojangles', 'C', 'Little Joey', '07/11/1991');
1 row inserted/updated/deleted

splice> INSERT INTO Players
    VALUES (99, 'Giants', 'Joe Bojangles', 'C', 'Little Joey', '07/11/1991'),
            (73, 'Giants', 'Lester Johns', 'P', 'Big John', '06/09/1984'),
            (27, 'Cards', 'Earl Hastings', 'OF', 'Speedy Earl', '04/22/1982');
3 rows inserted/updated/deleted
```

This example creates a table name OldGuys that has the same columns as our Players table, and then loads that table with the data from Players for all players born before 1980:

```
splice> CREATE TABLE OldGuys(
    ID          SMALLINT NOT NULL PRIMARY KEY,
    Team        VARCHAR(64) NOT NULL,
    Name        VARCHAR(64) NOT NULL,
    Position    CHAR(2),
    DisplayName VARCHAR(24),
    BirthDate   DATE
);

splice> INSERT INTO OldGuys
    SELECT * FROM Players
    WHERE BirthDate < '01/01/1980';
```

Statement dependency system

The `INSERT` statement depends on the table being inserted into, all of the conglomerates (units of storage such as heaps or indexes) for that table, and any other table named in the statement. Any statement that creates or drops an index or a constraint for the target table of a prepared `INSERT` statement invalidates the prepared `INSERT` statement.

See Also

- » [FETCH FIRST clause](#)
- » [ORDER BY clause](#)
- » [Queries](#)

» **RESULT OFFSET** clause

PIN TABLE

The `PIN TABLE` statement allows you to pin (cache) a table in memory, which can improve performance for tables that are being used frequently in analytic operations.

The pinned version of a table is a static version of that table; updates to the underlying table are not automatically reflected in the pinned version. To refresh the pinned version, you need to unpin and then repin the table, as described in the [Usage Notes](#) section below.

Syntax

```
PIN TABLE tableName;
```

tableName

A string that specifies the name of the table that you want to pin in memory.

An error occurs if the named table does not exist.

Usage Notes

Here are a few important notes about using pinned tables:

- » [Pinned and Unpinned Table Versions](#)
- » [Refreshing the Pinned Version of a Table](#)
- » [Unpinning and Dropping Pinned Tables](#)

Pinned and Unpinned Table Versions

Once you pin a table, you effectively have two versions of it to work with:

- » The original table continues to work just as usual
- » The pinned version is a static version of the table at the time you pinned it. To access the pinned version of a table, you must specify the Splice `pin=true` property. If you do not specify this property in your query, the query will operate on the unpinned version of the table.

The pinned version (version) of a table is statically cached in memory; this means that:

- » Updates to the table (unpinned version) are not automatically reflected in the pinned version of the table.
- » Updates to the pinned version of the table are not permitted: you cannot insert into, delete from, or update the pinned version of a table.

Here's a simple example that illustrates these qualities:

```

splice> CREATE TABLE myTbl (col1 int, col2 varchar(10));
0 rows inserted/updated/deleted
splice> INSERT INTO myTbl VALUES (1, 'One'), (2, 'Two');
2 rows inserted/updated/deleted
splice> PIN TABLE myTbl;
0 rows inserted/updated/deleted
splice> INSERT INTO myTbl VALUES (3, 'Three'), (4, 'Four');
2 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
COL1      | COL2
-----
1          One
2          Two
3          Three
4          Four

4 rows selected
splice> SELECT * FROM myTbl --splice-properties pin=true
> ;
COL1      | COL2
-----
1          One
2          Two
2 rows selectedsplice> UPDATE myTbl SET col1=11 WHERE col1=1;1 row inserted/updated/deletedsplice> UPDATE myTbl --splice-properties pin=trueSET col1=21 WHERE col1=2;ERROR: Pinned Table read failed with exception Table or view not found in database.

```

Refreshing the Pinned Version of a Table

If you update the table and want the pinned version to reflect those updates, you need to refresh your pinned table version. You can simply unpin the table from memory, and then repin it into memory:

```

splice> UNPIN TABLE Players;0 rows inserted/updated/deletedsplice> PIN TABLE Player
s;0 rows inserted/updated/deleted

```

Now the pinned version of the table matches the original version.

Unpinning and Dropping Pinned Tables

When you drop a table (with the `DROP TABLE` statement), the pinned version is automatically deleted and can no longer be used.

To delete just the pinned version of a table, use the `UNPIN TABLE` statement.

See Also

» [CREATE EXTERNAL TABLE](#) statement

- » [CREATE TABLE statement](#)
- » [UNPIN TABLE statement](#)
- » [Query Optimizations](#) in the *Splice Machine Developer's Guide*

RENAME COLUMN

Use the `RENAME COLUMN` statement to rename a column in a table.

The `RENAME COLUMN` statement allows you to rename an existing column in an existing table in any schema (except the schema `SYS`).

Syntax

```
RENAME COLUMN simple-Column-Name
    TO simple-Column-Name
```

table-Name

The name of the table containing the column to rename.

simple-Column-Name

The name of the column to be renamed.

simple-Column-Name

The new name for the column.

Usage Notes

To rename a column, you must either be the database owner or the table owner.

To perform other table alterations, see the [ALTER TABLE](#) statement.

If a view, trigger, check constraint, or *generation-clause* of a generated column references the column, an attempt to rename it will generate an error.

NOTE: The `RENAME COLUMN` statement is not allowed if there are any open cursors that reference the column that is being altered.

NOTE: If there is an index defined on the column, the column can still be renamed; the index is automatically updated to refer to the column by its new name.

Examples

To rename the `Birthdate` column in table `Players` to `BornDate`, use the following syntax:

```
splice> RENAME COLUMN Players.Birthdate TO BornDate;
0 rows inserted/updated/deleted
```

If you want to modify a column's data type, you can combine ALTER TABLE, UPDATE, and RENAME COLUMN using these steps, as shown in the example below:

1. Add a new column to the table with the new data type
2. Copy the values from the “old” column to the new column with an UPDATE statement.
3. Drop the “old” column.
4. Rename the new column with the old column’s name.

```
splice> ALTER TABLE Players ADD COLUMN NewPosition VARCHAR(8);
0 rows inserted/updated/deleted

splice> UPDATE Players SET NewPosition = Position;
0 rows inserted/updated/deleted

splice> ALTER TABLE Players DROP COLUMN Position;
0 rows inserted/updated/deleted

splice> RENAME COLUMN Players.NewPosition TO Position;
0 rows inserted/updated/deleted
```

See Also

- » [ALTER statement](#)

RENAME INDEX

The `RENAME INDEX` statement allows you to rename an index in the current schema. Users cannot rename indexes in the `SYS` schema.

Syntax

```
RENAME INDEX index-Name TO new-index-Name
```

index-Name

The name of the index to be renamed.

new-Index-Name

The new name for the index.

Example

```
splice> RENAME INDEX myIdx TO Player_index;  
0 rows inserted/updated/deleted
```

See Also

» [ALTER statement](#)

RENAME TABLE

The `RENAME TABLE` statement allows you to rename an existing table in any schema (except the schema `SYS`).

To rename a table, you must either be the database owner or the table owner.

Syntax

```
RENAME TABLE table-Name TO new-Table-Name
```

table-Name

The name of the table to be renamed.

new-Table-Name

The new name for the table.

Usage Notes

Attempting to rename a table generates an error if:

- » there is a view or a foreign key that references the table
- » there are any check constraints or triggers on the table

Example

```
splice> RENAME TABLE MorePlayers to PlayersTest;  
0 rows inserted/updated/deleted
```

See the [ALTER TABLE](#) statement for more information.

REVOKE

Use the REVOKE statement to remove privileges from a specific user or role, or from all users, to perform actions on database objects. You can also use the REVOKE statement to revoke a role from a user, from PUBLIC, or from another role.

The syntax that you use for the REVOKE statement depends on whether you are revoking privileges to a schema object or revoking a role.

Syntax for SCHEMA

```
REVOKE privilege-type
  ON [ SCHEMA ] schema
  FROM grantees
```

privilege-type

```
DELETE
| INSERT
| REFERENCES [( column-identifier {, column-identifier}* )]
| SELECT [( column-identifier {, column-identifier}* )]
| TRIGGER
| UPDATE [( column-identifier {, column-identifier}* )]
```

See the [Privilege Types](#) section below for more information.



Column-level privileges are available only with a Splice Machine Enterprise license.

You cannot grant or revoke privileges at the `column-identifier` level with the Community version of Splice Machine.

To obtain a license for the Splice Machine Enterprise Edition, please [Contact Splice Machine Sales](#) today.

schema-Name

The name of the schema for which you are revoking access.

grantees

The user(s) or role(s) for whom you are revoking access. See the [About Grantees](#) section below for more information.

Syntax for Tables

```
REVOKE privilege-type
  ON [ TABLE ] table-Name
  FROM grantees
```

privilege-type

```
DELETE
| INSERT
| REFERENCES [( column-identifier {, column-identifier}* )]
| SELECT [( column-identifier {, column-identifier}* )]
| TRIGGER
| UPDATE [( column-identifier {, column-identifier}* )]
```

See the [Privilege Types](#) section below for more information.



Column-level privileges are available only with a Splice Machine Enterprise license.

You cannot grant or revoke privileges at the `column-identifier` level with the Community version of Splice Machine.

To obtain a license for the Splice Machine Enterprise Edition, please [Contact Splice Machine Sales](#) today.

table-Name

The name of the table for which you are revoking access.

view-Name

The name of the view for which you are revoking access.

grantees

The user(s) or role(s) for whom you are revoking access. See the [About Grantees](#) section below for more information.

Syntax for Routines

```
REVOKE EXECUTE ON { FUNCTION | PROCEDURE }
  {function-name | procedure-name}
  TO grantees RESTRICT
```

function-name | procedure-name

The name of the function or procedure for which you are revoking access.

grantees

The user(s) or role(s) for whom you are revoking access. See the [About Grantees](#) section below for more information.

RESTRICT

You **must** use this clause when revoking access for routines.

The RESTRICT clause specifies that the EXECUTE privilege cannot be revoked if the specified routine is used in a view, trigger, or constraint, and the privilege is being revoked from the owner of the view, trigger, or constraint.

Syntax for User-defined types

```
REVOKE USAGE
  ON TYPE SQL Identifier
  FROM grantees RESTRICT
```

[schema-name.] SQL Identifier

The user-defined type (UDT) name is composed of an optional *schemaName* and a *SQL Identifier*. If a *schemaName* is not provided, the current schema is the default schema. If a qualified UDT name is specified, the schema name cannot begin with SYS.

grantees

The user(s) or role(s) for whom you are revoking access. See the [About Grantees](#) section below for more information.

RESTRICT

You **must** use this clause when revoking access for user-defined types.

The RESTRICT clause specifies that the EXECUTE privilege cannot be revoked if the specified UDT is used in a view, trigger, or constraint, and the privilege is being revoked from the owner of the view, trigger, or constraint.

Syntax for Roles

```
REVOKE roleName
  { roleName }*
  FROM grantees
```

roleName

The name to the role(s) for which you are revoking access.

grantees

The user(s) or role(s) for whom you are revoking access. See the [About Grantees](#) section below for more information.

Only the database owner can revoke a role.

About Grantees

A grantee can be one or more specific users, one or more specific roles, or all users (`PUBLIC`). Either the object owner or the database owner can grant privileges to a user or to a role. Only the database owner can grant a role to a user or to another role.

Here's the syntax:

```
{      roleName | PUBLIC }
[, { roleName | PUBLIC } ] *
```

AuthorizationIdentifier

An expression.

roleName

The name of the role.

Either the object owner or the database owner can grant privileges to a user or to a role. Only the database owner can grant a role to a user or to another role.

PUBLIC

Use the keyword `PUBLIC` to specify all users.

When `PUBLIC` is specified, the privileges or roles affect all current and future users.

The privileges granted to `PUBLIC` and to individual users or roles are independent privileges. For example, a `SELECT` privilege on table `t` is granted to both `PUBLIC` and to the authorization ID `harry`. If the `SELECT` privilege is later revoked from the authorization ID `harry`, Harry will still be able to access the table `t` through the `PUBLIC` privilege..

Privilege Types

Privilege Type	Usage
ALL PRIVILEGES	To revoke all of the privileges to the user or role for the specified table. You can also grant one or more table privileges by specifying a privilege-list.
DELETE	To revoke permission to delete rows from the specified table.
INSERT	To revoke permission to insert rows into the specified table.
REFERENCES	To revoke permission to create a foreign key reference to the specified table. If a column list is specified with the REFERENCES privilege, the permission is valid on only the foreign key reference to the specified columns.

Privilege Type	Usage
SELECT	To revoke permission to perform SelectExpressions on a table or view. If a column list is specified with the SELECT privilege, the permission is valid on only those columns. If no column list is specified, then the privilege is valid on all of the columns in the table. For queries that do not select a specific column from the tables involved in a SELECT statement or <i>SelectExpression</i> (for example, queries that use COUNT(*)), the user must have at least one column-level SELECT privilege or table-level SELECT privilege.
TRIGGER	To revoke permission to create a trigger on the specified table.
UPDATE	To revoke permission to use the WHERE clause, you must have the SELECT privilege on the columns in the row that you want to update.

Usage Notes

The following types of privileges can be revoked:

- » Delete data from a specific table.
- » Insert data into a specific table.
- » Create a foreign key reference to the named table or to a subset of columns from a table.
- » Select data from a table, view, or a subset of columns in a table.
- » Create a trigger on a table.
- » Update data in a table or in a subset of columns in a table.
- » Run a specified routine (function or procedure).
- » Use a user-defined type.

Before you issue a REVOKE statement, check that the derby.database.sqlAuthorization property is set to true. The derby.database.sqlAuthorization property enables the SQL Authorization mode.

You can revoke privileges on an object if you are the owner of the object or the database owner. See the CREATE statement for the database object that you want To revoke privileges on for more information.

You can revoke privileges for an object if you are the owner of the object or the database owner.

Prepared statements and open result sets

Checking for privileges happens at statement execution time, so prepared statements are still usable after a revoke action. If sufficient privileges are still available for the session, prepared statements will be executed, and for queries, a result set will be returned.

Once a result set has been returned to the application (by executing a prepared statement or by direct execution), it will remain accessible even if privileges or roles are revoked in a way that would cause another execution of the same statement to fail.

Cascading object dependencies

For views, triggers, and constraints, if the privilege on which the object depends on is revoked, the object is automatically dropped. Splice Machine does not try to determine if you have other privileges that can replace the privileges that are being revoked.

Limitations

The following limitations apply to the REVOKE statement:

Table-level privileges

All of the table-level privilege types for a specified grantee and table ID are stored in one row in the SYSTABLEPERMS system table. For example, when user2 is granted the SELECT and DELETE privileges on table user1.t1, a row is added to the SYSTABLEPERMS table. The GRANTEE field contains user2 and the TABLEID contains user1.t1. The SELECTPRIV and DELETEPRIV fields are set to Y. The remaining privilege type fields are set to N.

When a grantee creates an object that relies on one of the privilege types, Splice Machine engine tracks the dependency of the object on the specific row in the SYSTABLEPERMS table. For example, user2 creates the view v1 by using the statement `SELECT * FROM user1.t1`, the dependency manager tracks the dependency of view v1 on the row in SYSTABLEPERMS for GRANTEE(user2), TABLEID(user1.t1).

The dependency manager knows only that the view is dependent on a privilege type in that specific row, but does not track exactly which privilege type the view is dependent on.

When a REVOKE statement for a table-level privilege is issued for a grantee and table ID, all of the objects that are dependent on the grantee and table ID are dropped. For example, if user1 revokes the DELETE privilege on table t1 from user2, the row in SYSTABLEPERMS for GRANTEE(user2), TABLEID(user1.t1) is modified by the REVOKE statement. The dependency manager sends a revoke invalidation message to the view user2.v1 and the view is dropped even though the view is not dependent on the DELETE privilege for GRANTEE(user2), TABLEID(user1.t1).

Column-level privileges

Only one type of privilege for a specified grantee and table ID are stored in one row in the SYS_COLPERMS system table. For example, when user2 is granted the SELECT privilege on table user1.t1 for columns c12 and c13, a row is added to the SYS_COLPERMS. The GRANTEE field contains user2, the TABLEID contains user1.t1, the TYPE field contains S, and the COLUMNS field contains c12, c13.

When a grantee creates an object that relies on the privilege type and the subset of columns in a table ID, Splice Machine engine tracks the dependency of the object on the specific row in the SYSCOLPERMS table. For example, user2 creates the view v1 by using the statement `SELECT c11 FROM user1.t1`, the dependency manager tracks the dependency of view v1 on the row in SYSCOLPERMS for GRANTEE(user2), TABLEID(user1.t1), TYPE(s). The dependency manager knows that the view is dependent on the SELECT privilege type, but does not track exactly which columns the view is dependent on.

When a REVOKE statement for a column-level privilege is issued for a grantee, table ID, and type, all of the objects that are dependent on the grantee, table ID, and type are dropped. For example, if user1 revokes the SELECT privilege on column c12 on table user1.t1 from user2, the row in SYSCOLPERMS for GRANTEE(user2), TABLEID(user1.t1), TYPE(s) is modified by the REVOKE statement. The dependency manager sends a revoke invalidation message to the view user2.v1 and the view is dropped even though the view is not dependent on the column c12 for GRANTEE(user2), TABLEID(user1.t1), TYPE(s).

Roles

Splice Machine tracks any dependencies on the definer's current role for views and constraints, constraints, and triggers. If privileges were obtainable only via the current role when the object in question was defined, that object depends on the current role. The object will be dropped if the role is revoked from the defining user or from PUBLIC, as the case may be.

Also, if a contained role of the current role in such cases is revoked, dependent objects will be dropped. Note that dropping may be too pessimistic. This is because Splice Machine does not currently make an attempt to recheck if the necessary privileges are still available in such cases.

Revoke Examples

Revoking User Privileges

To revoke the SELECT privilege on schema SpliceBBall from the authorization IDs Bill and Joan, use the following syntax:

```
splice> REVOKE SELECT ON SCHEMA SpliceBBall FROM Bill, Joan;
0 rows inserted/updated/deleted
```

To revoke the SELECT privilege on table Salaries from the authorization IDs Bill and Joan, use the following syntax:

```
splice> REVOKE SELECT ON TABLE Salaries FROM Bill, Joan;
0 rows inserted/updated/deleted
```

To revoke the UPDATE and TRIGGER privileges on table Salaries from the authorization IDs Joe and Anita, use the following syntax:

```
splice> REVOKE UPDATE, TRIGGER ON TABLE Salaries FROM Joe, Anita;
0 rows inserted/updated/deleted
```

To revoke the SELECT privilege on table Hitting in the Baseball_stats schema from all users, use the following syntax:

```
splice> REVOKE SELECT ON TABLE Baseball_Stats.Hitting FROM PUBLIC;
0 rows inserted/updated/deleted
```

To revoke the EXECUTE privilege on procedure ComputeValue from the authorization ID george, use the following syntax:

```
splice> REVOKE EXECUTE ON PROCEDURE ComputeValue FROM george;
0 rows inserted/updated/deleted
```

Revoking User Roles

To revoke the role `purchases_reader_role` from the authorization IDs george and maria, use the following syntax:

```
splice> REVOKE purchases_reader_role FROM george,maria;
0 rows inserted/updated/deleted
```

Revoking Role Privileges

To revoke the SELECT privilege on schema SpliceBBall from the role purchases_reader_role, use the following syntax:

```
splice> REVOKE SELECT ON SCHEMA SpliceBBall FROM purchases_reader_role;
0 rows inserted/updated/deleted
```

To revoke the SELECT privilege on table t to the role purchases_reader_role, use the following syntax:

```
splice> REVOKE SELECT ON TABLE t FROM purchases_reader_role;
0 rows inserted/updated/deleted
```

See Also

- » [CREATE ROLE statement](#)
- » [DROP_ROLE statement](#)
- » [GRANT statement](#)
- » [RoleName](#)
- » [SET ROLE statement](#)
- » [SELECT expression](#)
- » [SELECT statement](#)
- » [SYSROLES system table](#)
- » [UPDATE statement](#)

» WHERE clause

SELECT

Use the SELECT statement to query a database and receive back results.

Syntax

```
SELECT Query
  [ORDER BY clause]
  [result offset clause]
  [fetch first clause]
```

Query

The SELECT statement is so named because the typical first word of the query construct is SELECT. (*Query* includes the [SELECT](#) expressions).

ORDER BY clause

The [ORDER BY](#) clause allows you to order the results of the SELECT. Without the ORDER BY clause, the results are returned in random order.

result offset and fetch first clauses

The [fetch first](#) clause, which can be combined with the [result offset](#) clause, limits the number of rows fetched.

Example

This example selectss all records in the Players table:

```
splice> SELECT * FROM Players WHERE ID < 11;
ID | TEAM      | POS&|DISPLAYNAME          | BIRTHDATE
---+-----+-----+-----+-----+
1  | Giants    | C   | Buddy Painter        | 1987-03-27
2  | Giants    | 1B  | Billy Bopper         | 1988-04-20
3  | Giants    | 2B  | John Purser          | 1990-10-30
4  | Giants    | SS  | Bob Cranker          | 1987-01-21
5  | Giants    | 3B  | Mitch Duffer          | 1991-01-15
6  | Giants    | LF  | Norman Aikman        | 1982-01-05
7  | Giants    | CF  | Alex Paramour        | 1981-07-02
8  | Giants    | RF  | Harry Pennello        | 1983-04-13
9  | Giants    | OF  | Greg Brown            | 1983-12-24
10 | Giants   | RF  | Jason Minman          | 1983-11-06
10 rows selected
```

This example selects the Birthdate of all players born in November or December:

```
splice> SELECT BirthDate
      FROM Players
     WHERE MONTH(BirthDate) > 10
     ORDER BY BIRTHDATE;
```

BIRTHDATE

```
-----
1980-12-19
1983-11-06
1983-11-28
1983-12-24
1984-11-22
1985-11-07
1985-11-26
1985-12-21
1986-11-13
1986-11-24
1986-12-16
1987-11-12
1987-11-16
1987-12-17
1988-12-21
1989-11-17
1991-11-15
```

17 rows selected

This example selects the name, team, and birth date of all players born in 1985 and 1989:

```
splice> SELECT DisplayName, Team, BirthDate
      FROM Players
     WHERE YEAR(BirthDate) IN (1985, 1989)
     ORDER BY BirthDate;
```

DISPLAYNAME	TEAM	BIRTHDATE
Jeremy Johnson	Cards	1985-03-15
Gary Kosovo	Giants	1985-06-12
Michael Hillson	Cards	1985-11-07
Mitch Canepa	Cards	1985-11-26
Edward Erdman	Cards	1985-12-21
Jeremy Packman	Giants	1989-01-01
Nathan Nickels	Giants	1989-05-04
Ken Straiter	Cards	1989-07-20
Marcus Bamburger	Giants	1989-08-01
George Goomba	Cards	1989-08-08
Jack Hellman	Cards	1989-08-09
Elliot Andrews	Giants	1989-08-21
Henry Socomy	Giants	1989-11-17

13 rows selected

Statement dependency system

The SELECT statement depends on all the tables and views named in the query and the conglomerates (units of storage such as heaps and indexes) chosen for access paths on those tables.

The SELECT statement depends on all aliases used in the query. Dropping an alias invalidates any prepared SELECT statement that uses the alias.

See Also

- » [CREATE INDEXstatement](#)
- » [CREATE VIEWstatement](#)
- » [DROP INDEXstatement](#)
- » [DROP VIEWstatement](#)
- » [GRANTstatement](#)
- » [ORDER BYclause](#)
- » [FETCH FIRSTclause](#)
- » [RESULT OFFSETclause](#)

SET ROLE

The `SET ROLE` statement allows you to set the current role for the current SQL context of a session.

You can set a role only if the current user has been granted the role, or if the role has been granted to `PUBLIC`.

NOTE: The `SET ROLE` statement is not transactional; a rollback does not undo the effect of setting a role. If a transaction is in progress, an attempt to set a role results in an error.

Syntax

```
SET ROLE { roleName | 'string-constant' | ? | NONE }
```

roleName

The role you want set as the current role.

You can specify a *roleName* of `NONE` to unset the current role.

If you specify the role as a string constant or as a dynamic parameter specification (?), any leading and trailing blanks are trimmed from the string before attempting to use the remaining (sub)string as a *roleName*. The dynamic parameter specification can be used in prepared statements, so the `SET ROLE` statement can be prepared once and then executed with different role values. You cannot specify `NONE` as a dynamic parameter.

Usage Notes

Setting a role identifies a set of privileges that is a union of the following:

- » The privileges granted to that role
- » The union of privileges of roles contained in that role (for a definition of role containment, see “Syntax for roles” in [GRANT statement](#))

In a session, the *current privileges* define what the session is allowed to access. The *current privileges* are the union of the following:

- » The privileges granted to the current user
- » The privileges granted to `PUBLIC`
- » The privileges identified by the current role, if set

NOTE: You can find the available role names in the `SYS.SYSROLES` system table.

SQL Example

This examples set the role of the current user to `reader_role`:

```
splice> SET ROLE reader_role;
0 rows inserted/updated/deleted
```

JDBC Example

This examples set the role of the current user to `reader_role`:

```
stmt.execute("SET ROLE admin");      -- case normal form: ADMIN
stmt.execute("SET ROLE \"admin\"");   -- case normal form: admin
stmt.execute("SET ROLE none");       -- special case

PreparedStatement ps = conn.prepareStatement("SET ROLE ?");
ps.setString(1, " admin ");        -- on execute: case normal form: ADMIN
ps.setString(1, "\\"admin\\\"");     -- on execute: case normal form: admin
ps.setString(1, "none");           -- on execute: syntax error
ps.setString(1, "\\\"none\\\"");     -- on execute: case normal form: none
```

See Also

- » [CREATE_ROLE statement](#)
- » [DROP_ROLE statement](#)
- » [GRANT statement](#)
- » [REVOKE statement](#)
- » [RoleName](#)
- » [SET_ROLE statement](#)
- » [SELECT expression](#)
- » [SELECT statement](#)
- » [SYSROLES system table](#)
- » [UPDATE statement](#)
- » [WHERE clause](#)

SET SCHEMA

The `SET SCHEMA` statement sets the default schema for a connection's session to the designated schema. The default schema is used as the target schema for all statements issued from the connection that do not explicitly specify a schema name.

The target schema must exist for the `SET SCHEMA` statement to succeed. If the schema doesn't exist an error is returned.

NOTE: The `SET SCHEMA` statement is not transactional: if the `SET SCHEMA` statement is part of a transaction that is rolled back, the schema change remains in effect.

Syntax

```
SET [CURRENT] SCHEMA [=] schemaName
```

schemaName

The name of the schema; this name is not case sensitive.

Examples

These examples are equivalent:

```
splice> SET SCHEMA BASEBALL;
0 rows inserted/updated/deleted

splice> SET SCHEMA Baseball;
0 rows inserted/updated/deleted

splice> SET CURRENT SCHEMA BaseBall;
0 rows inserted/updated/deleted

splice> SET CURRENT SQLID BASEBALL;
0 rows inserted/updated/deleted

splice> SET SCHEMA "BASEBALL";
0 rows inserted/updated/deleted

splice> SET SCHEMA 'BASEBALL'
0 rows inserted/updated/deleted
```

These fail because of case sensitivity:

```
splice> SET SCHEMA "Baseball";
ERROR 42Y07: Schema 'Baseball' does not exist

splice> SET SCHEMA 'BaseBall';
ERROR 42Y07: Schema 'BaseBall' does not exist
```

Here's an example using a prepared statement:

```
PreparedStatement ps = conn.prepareStatement("set schema ?");
ps.setString(1,"HOTEL");
ps.executeUpdate();
... these work:
ps.setString(1,"SPLICE");

ps.executeUpdate();
ps.setString(1,"splice");           //error - string is case sensitive
// no app will be found
ps.setNull(1, Types.VARCHAR); //error - null is not allowed
```

See Also

- » [CREATE SCHEMA statement](#)
- » [DROP SCHEMA statement](#)
- » [Schema Name](#)

TRUNCATE TABLE

The TRUNCATE TABLE statement allows you to quickly remove all content from the specified table and return it to its initial empty state.

To truncate a table, you must either be the database owner or the table owner.

You cannot truncate system tables or global temporary tables with this statement.

Syntax

```
TRUNCATE TABLE table-Name
```

table-Name

The name of the table to truncate.

Examples

To truncate the entire `Players_Test` table, use the following statement:

```
splice> TRUNCATE TABLE Players_Test;
0 rows inserted/updated/deleted
```

UNPIN TABLE

The UNPIN TABLE statement unpins a table, which means that the pinned (previously cached) version of the table no longer exists.

Syntax

```
UNPIN TABLE table-Name
```

table-Name

The name of the pinned table that you want to unpin.

Example

```
splice> CREATE TABLE myTbl (col1 int, col2 varchar(10));
0 rows inserted/updated/deleted
splice> INSERT INTO myTbl VALUES (1, 'One'), (2, 'Two');
2 rows inserted/updated/deleted
COL 1      |COL2
-----
1          One
2          Two
2 rows selected
splice> PIN TABLE myTbl;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl --splice-properties pin=true
> ;
COL 1      |COL2
-----
1          One
2          Two
2 rows selected
splice> UNPIN TABLE myTbl;splice> SELECT * FROM myTbl;
COL 1      |COL2
-----
1          One
2          Two
2 rows selected
splice> SELECT * FROM myTbl --splice-properties pin=true
> ERROR: Pinned table read failed with exception 'Table or view not found in database'
```

See Also

- » [CREATE EXTERNAL TABLE statement](#)
- » [CREATE TABLE statement](#)
- » [PIN TABLE statement](#)
- » [Query Optimizations](#) in the *Splice Machine Developer's Guide*

UPDATE

Use the UPDATE statement to update existing records in a table.

Syntax

```
{
  UPDATE table-Name
    [ [AS] correlation-Name ]
    SET column-Name = Value
        [ , column-Name = Value } ]*
    [ WHERE clause ]
}
```

table-Name

The name of the table to update.

correlation-Name

An optional correlation name for the update.

column-Name = *Value*

Sets the value of the named column to the named value in any records .

Value is either an *Expression* or the literal DEFAULT. If you specify DEFAULT for a column's value, the value is set to the default defined for the column in the table.

The DEFAULT literal is the only value that you can directly assign to a generated column. Whenever you alter the value of a column referenced by the *generation-clause* of a generated column, Splice Machine recalculates the value of the generated column.

WHERE clause

Specifies the records to be updated.

Example

This example updates the Birthdate value for a specific player:

```
splice> UPDATE Players
      SET Birthdate='03/27/1987'
      WHERE DisplayName='Buddy Painter';
1 row inserted/updated/deleted
```

This example updates the team name associated with all players on the Giants team:

```
splice> UPDATE Players
      SET Team='SFGiants'
      WHERE Team='Giants';
48 rows inserted/updated/deleted
```

Statement dependency system

A searched update statement depends on the table being updated, all of its conglomerates (units of storage such as heaps or indexes), all of its constraints, and any other table named in the `DROP INDEX` statement or an `ALTER TABLE` statement for the target table of a prepared searched update statement invalidates the prepared searched update statement.

A `CREATE` or `DROP INDEX` statement or an `ALTER TABLE` statement for the target table of a prepared positioned update invalidates the prepared positioned update statement.

Dropping an alias invalidates a prepared update statement if the latter statement uses the alias.

Dropping or adding triggers on the target table of the update invalidates the update statement.

See Also

- » [ALTER TABLEstatement](#)
- » [CONSTRAINTclause](#)
- » [CREATE TABLEstatement](#)
- » [CREATE TRIGGER](#)
- » [DROP INDEXstatement](#)
- » [DROP TRIGGER](#)
- » [WHEREclause](#)

Clauses

This section contains the reference documentation for the Splice Machine SQL Clauses, in the following topics:

Clause	Description
CONSTRAINT	Optional clause in ALTER TABLE statements that specifies a rule to which the data must conform.
EXCEPT	Takes the distinct rows in the results from one a SELECT statement.
FROM	A clause in a <i>SelectExpression</i> that specifies the tables from which the other clauses of the query can access columns for use in expressions.
GROUP BY	Part of a <i>SelectExpression</i> that groups a result into subsets that have matching values for one or more columns.
HAVING	Restricts the results of a GROUP BY clause in a <i>SelectExpression</i> .
LIMIT n	Limits the number of results returned by a query.
OVER	Used in window functions to define the window on which the function operates.
ORDER BY	Allows you to specify the order in which rows appear in the result set.
RESULT OFFSET and FETCH FIRST	Provide a way to skip the N first rows in a result set before starting to return any rows and/or to limit the number of rows returned in the result set.
TOP n	Limits the number of results returned by a query.
UNION	Combines the result sets from two queries into a single table that contains all matching rows.
USING	Specifies which columns to test for equality when two tables are joined.
WHERE	An optional part of a UPDATE statement that lets you select rows based on a Boolean expression.
WITH	Allows you to name subqueries to make your queries more readable and/or to improve efficiency.



For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](#).

CONSTRAINT

A CONSTRAINT clause is a rule to which data must conform, and is an optional part of `ALTER TABLE` statements. Constraints can optionally be named.

There are two types of constraints:

column-level constraints

A column-level constraint refers to a single column in a table (the column that it follows syntactically) in the table. Column constraints, other than CHECK constraints, do not specify a column name.

table-level constraints

A table-level constraints refers to one or more columns in a table by specifying the names of those columns. Table-level CHECK constraints can refer to 0 or more columns in the table.

Column constraints and table constraints have the same function; the difference is in where you specify them.

- » Table constraints allow you to specify more than one column in a PRIMARY KEY or CHECK , UNIQUE or FOREIGN KEY constraint definition.
- » Column-level constraints (except for check constraints) refer to only one column.

Column Constraints

```
{
  NOT NULL |
  [ [CONSTRAINT constraint-Name] {PRIMARY KEY} ]
}
```

```
{
  NOT NULL |
  [ [CONSTRAINT constraint-Name]
  {
    CHECK (searchCondition) |
    {
      PRIMARY KEY |
      UNIQUE |
      REFERENCES clause
    }
  }
}
```

NOT NULL

Specifies that this column cannot hold NULL values (constraints of this type are not nameable).

PRIMARY KEY

Specifies the column that uniquely identifies a row in the table. The identified columns must be defined as NOT NULL.

NOTE: At this time, you **cannot** add a primary key using ALTER TABLE.

UNIQUE

Specifies that values in the column must be unique.

FOREIGN KEY

Specifies that the values in the column must correspond to values in a referenced primary key or unique key column or that they are NULL.

CHECK

Specifies rules for values in the column.

Table Constraints

```
[CONSTRAINT constraint-Name]
{
    PRIMARY KEY ( Simple-column-Name
    [ , Simple-column-Name ]* )
}
```

```
[CONSTRAINT constraint-Name]
{
    CHECK (searchCondition) |
    {
        PRIMARY KEY ( Simple-column-Name [ , Simple-column-Name ]* ) |
        UNIQUE ( Simple-column-Name [ , Simple-column-Name ]* ) |
        FOREIGN KEY ( Simple-column-Name [ , Simple-column-Name ]* )
        REFERENCES clause
    }
}
```

PRIMARY KEY

Specifies the column or columns that uniquely identify a row in the table. NULL values are not allowed.

NOTE: At this time, you **cannot** add a primary key using ALTER TABLE.

UNIQUE

Specifies that values in the columns must be unique.

FOREIGN KEY

Specifies that the values in the columns must correspond to values in referenced primary key or unique columns or that they are NULL.

NOTE: If the foreign key consists of multiple columns, and *any* column is NULL, the whole key is considered NULL. The insert is permitted no matter what is on the non-null columns.

CHECK

Specifies a wide range of rules for values in the table.

Primary Key Constraints

NOTE: At this time, you **cannot** alter primary keys using ALTER TABLE.

Primary keys are constrained as follows:

- » A primary key defines the set of columns that uniquely identifies rows in a table.
- » When you create a primary key constraint, none of the columns included in the primary key can have NULL constraints; that is, they must not permit NULL values.
- » A table can have at most one PRIMARY KEY constraint.

Unique constraints

A UNIQUE constraint defines a set of columns that uniquely identify rows in a table only if all the key values are not NULL. If one or more key parts are NULL, duplicate keys are allowed.

For example, if there is a UNIQUE constraint on col1 and col2 of a table, the combination of the values held by col1 and col2 will be unique as long as these values are not NULL. If one of col1 and col2 holds a NULL value, there can be another identical row in the table.

A table can have multiple UNIQUE constraints.

Foreign key constraints

Foreign keys provide a way to enforce the referential integrity of a database. A foreign key is a column or group of columns within a table that references a key in some other table (or sometimes the same table). The foreign key must always include the columns of which the types exactly match those in the referenced primary key or unique constraint.

For a table-level foreign key constraint in which you specify the columns in the table that make up the constraint, you cannot use the same column more than once.

If there is a column list in the *ReferencesSpecification* (a list of columns in the referenced table), it must correspond either to a unique constraint or to a primary key constraint in the referenced table. The *ReferencesSpecification* can omit the column list for the referenced table if that table has a declared primary key.

If there is no column list in the *ReferencesSpecification* and the referenced table has no primary key, a statement exception is thrown. (This means that if the referenced table has only unique keys, you must include a column list in the *ReferencesSpecification*.)

A foreign key constraint is satisfied if there is a matching value in the referenced unique or primary key column. If the foreign key consists of multiple columns, the foreign key value is considered `NULL` if any of its columns contains a `NULL`.

It is possible for a foreign key consisting of multiple columns to allow one of the columns to contain a value for which there is no matching value in the referenced columns, per the ANSI SQL standard. To avoid this situation, create `NOT NULL` constraints on all of the foreign key's columns.

Foreign key constraints and DML

When you insert into or update a table with an enabled foreign key constraint, Splice Machine checks that the row does not violate the foreign key constraint by looking up the corresponding referenced key in the referenced table. If the constraint is not satisfied, Splice Machine rejects the insert or update with a statement exception.

When you update or delete a row in a table with a referenced key (a primary or unique constraint referenced by a foreign key), Splice Machine checks every foreign key constraint that references the key to make sure that the removal or modification of the row does not cause a constraint violation.

If removal or modification of the row would cause a constraint violation, the update or delete is not permitted and Splice Machine throws a statement exception.

Splice Machine performs constraint checks at the time the statement is executed, not when the transaction commits.

`PRIMARY KEY` constraints generate unique indexes. `FOREIGN KEY` constraints generate non-unique indexes.

`UNIQUE` constraints generate unique indexes if all the columns are non-nullable, and they generate non-unique indexes if one or more columns are nullable.

Therefore, if a column or set of columns has a `UNIQUE`, `PRIMARY KEY`, or `FOREIGN KEY` constraint on it, you do not need to create an index on those columns for performance. Splice Machine has already created it for you.

Check constraints

You can use check constraints to limit which values are accepted by one or more columns in a table. You specify the constraint with a Boolean expression; if the expression evaluates to `true`, the value is allowed; if the expression evaluates to `false`, the constraint prevents the value from being entered into the database. The search condition is applied to each row that is modified on an `INSERT` or `UPDATE` at the time of the row modification. When a constraint is violated, the entire statement is aborted. You can apply check constraints at the column level or table level.

For example, you could specify that values in the salary column for the players on your team must be between \$250,000 and \$30,000,000 with this expression:

```
salary >= 250000 AND salary <= 30000000.
```

Any attempt to insert or update a record with a salary value out of that range would fail.

Search Condition

A *searchCondition* is any Boolean expression that meets the requirements specified below. If a *constraint-Name* is not specified, Splice Machine generates a unique constraint name (for either column or table constraints).

Requirements for search condition

If a check constraint is specified as part of a column-definition, a column reference can only be made to the same column. Check constraints specified as part of a table definition can have column references identifying columns previously defined in the `CREATE TABLE` statement.

The search condition must always return the same value if applied to the same values. Thus, it cannot contain any of the following:

- » Dynamic parameters
- » Date/Time Functions (`CURRENT_TIMESTAMP`)
- » Subqueries
- » User Functions (such as `CURRENT_USER`)

Examples

```

-- column-level primary key constraint named OUT_TRAY_PK:
CREATE TABLE SAMP.OUT_TRAY
(
SENT TIMESTAMP,
DESTINATION CHAR(8),
SUBJECT CHAR(64) NOT NULL CONSTRAINT
OUT_TRAY_PK PRIMARY KEY,
NOTE_TEXT VARCHAR(3000)
);

-- the table-level primary key definition allows you to
-- include two columns in the primary key definition:
CREATE TABLE SAMP.SCHED
(
CLASS_CODE CHAR(7) NOT NULL,
DAY SMALLINT NOT NULL,
STARTING TIME,
ENDING TIME,
PRIMARY KEY (CLASS_CODE, DAY)
);
-- Use a column-level constraint for an arithmetic check
-- Use a table-level constraint
-- to make sure that a employee's taxes does not
-- exceed the bonus
CREATE TABLE SAMP.EMP
(
EMPNO CHAR(6) NOT NULL CONSTRAINT EMP_PK PRIMARY KEY,
FIRSTNME CHAR(12) NOT NULL,
MIDINIT VARCHAR(12) NOT NULL,
LASTNAME VARCHAR(15) NOT NULL,
SALARY DECIMAL(9,2) CONSTRAINT SAL_CK CHECK (SALARY >= 10000),
BONUS DECIMAL(9,2),
TAX DECIMAL(9,2),
CONSTRAINT BONUS_CK CHECK (BONUS > TAX)
);

-- use a check constraint to allow only appropriate
-- abbreviations for the meals
CREATE TABLE FLIGHTS
(
FLIGHT_ID CHAR(6) NOT NULL ,
SEGMENT_NUMBER INTEGER NOT NULL ,
ORIG_AIRPORT CHAR(3),
DEPART_TIME TIME,
DEST_AIRPORT CHAR(3),
ARRIVE_TIME TIME,
MEAL CHAR(1) CONSTRAINT MEAL_CONSTRAINT
CHECK (MEAL IN ('B', 'L', 'D', 'S')),
PRIMARY KEY (FLIGHT_ID, SEGMENT_NUMBER)
);

```

Statement dependency system

`INSERT` and `UPDATE` statements depend on all constraints on the target table.

`DELETE` statements depend on unique, primary key, and foreign key constraints.

These statements are invalidated if a constraint is added to or dropped from the target table.

See Also

- » [ALTER TABLE statement](#)
- » [CREATE TABLE statement](#)
- » [INSERT statement](#)
- » [DELETE statement](#)
- » [Foreign Keys](#) in the *Developer's Guide*.
- » [Triggers](#) in the *Developer's Guide*.
- » [UPDATE statement](#)

EXCEPT

The EXCEPT operator combines the result set of two or more similar SELECT queries, returning the results from the first query that do not appear in the results of the second query.

Syntax

```
EXCEPT [ SELECT expression ]*
```

SELECT expression

A SELECT expression that does not include an ORDER BY clause.

If you include an ORDER BY clause, that clause applies to the intersection operation.

DISTINCT

(Optional). Indicates that only distinct (non-duplicate) rows from the queries are included. This is the default.

ALL

(Optional). Indicates that all rows from the queries are included, including duplicates. With ALL, a row that has m duplicates in the left table and n duplicates in the right table will appear $\max(m-n, 0)$ times in the result set.

Usage

Each SELECT statement in the operation must contain the same number of columns, with similar data types, in the same order. Although the number, data types, and order of the fields in the select queries that you combine in an EXCEPT clause must correspond, you can use expressions, such as calculations or subqueries, to make them correspond.

When comparing column values for determining DISTINCT rows, two NULL values are considered equal.

Results

A result set.

Examples

```

CREATE TABLE t1( id INTEGER NOT NULL PRIMARY KEY,
                 i1 INTEGER, i2 INTEGER,
                 c10 char(10), c30 char(30), tm time);

CREATE TABLE t2( id INTEGER NOT NULL PRIMARY KEY,
                 i1 INTEGER, i2 INTEGER,
                 vc20 varchar(20), d double, dt date);

INSERT INTO t1(id,i1,i2,c10,c30) VALUES
  (1,1,1,'a','123456789012345678901234567890'),
  (2,1,2,'a','bb'),
  (3,1,3,'b','bb'),
  (4,1,3,'zz','5'),
  (5,NULL,NULL,NULL,'1.0'),
  (6,NULL,NULL,NULL,'a');

INSERT INTO t2(id,i1,i2,vc20,d) VALUES
  (1,1,1,'a',1.0),
  (2,1,2,'a',1.1),
  (5,NULL,NULL,'12345678901234567890',3),
  (100,1,3,'zz',3),
  (101,1,2,'bb',NULL),
  (102,5,5,'',NULL),
  (103,1,3,' a',NULL),
  (104,1,3,'NULL',7.4);

```

```

splice> SELECT id,i1,i2 FROM t1 EXCEPT SELECT id,i1,i2 FROM t2 ORDER BY id,i1,i2;
ID      | I1      | I2
-----
4      | 1      | 3
3      | 1      | 3
6      | NULL   | NULL

```

3 rows selected

```

splice> SELECT i1,i2 FROM t1 EXCEPT SELECT i1,i2 FROM t2 where id = -1 ORDER BY 1,2;
I1      | I2
-----
NULL    | NULL
1      | 3
1      | 2
1      | 1

```

4 rows selected

```
splice> SELECT i1,i2 FROM t1 where id = -1 EXCEPT SELECT i1,i2 FROM t2 ORDER BY 1,2;  
I1          | I2  
-----  
0 rows selected
```

See Also

- » Union clause

FROM

The FROM clause is a mandatory clause in a *SelectExpression*. It specifies the tables (*TableExpression*) from which the other clauses of the query can access columns for use in expressions.

Syntax

```
FROM TableExpression [ , TableExpression ]*
```

TableExpression

Specifies a table, view, or function; it is the source from which a *TableExpression* selects a result.

Examples

```
SELECT Cities.city_id
  FROM Cities
 WHERE city_id < 5;

-- other types of TableExpressions
SELECT TABLENAME, ISINDEX
  FROM SYS.SYSTABLES T, SYS.SYSCONGLOMERATES C
 WHERE T.TABLEID = C.TABLEID
 ORDER BY TABLENAME, ISINDEX;

-- force the join order
SELECT *
  FROM Flights, FlightAvailability
 WHERE FlightAvailability.flight_id = Flights.flight_id
   AND FlightAvailability.segment_number = Flights.segment_number
   AND Flights.flight_id < 'AA1115';

-- a TableExpression can be a joinOperation. Therefore
-- you can have multiple join operations in a FROM clause
SELECT COUNTRIES.COUNTRY, CITIES.CITY_NAME,
       FLIGHTS.DEST_AIRPORT
  FROM COUNTRIES LEFT OUTER JOIN CITIES
    ON COUNTRIES.COUNTRY_ISO_CODE = CITIES.COUNTRY_ISO_CODE
   LEFT OUTER JOIN FLIGHTS
     ON Cities.AIRPORT = FLIGHTS.DEST_AIRPORT;
```

See Also

» [SELECT expression](#)

» TABLE expression

GROUP BY

A GROUP BY clause is part of a *SelectExpression*, that groups a result into subsets that have matching values for one or more columns. In each group, no two rows have the same value for the grouping column or columns. NULLs are considered equivalent for grouping purposes.

You typically use a GROUP BY clause in conjunction with an aggregate expression.

Using the ROLLUP syntax, you can specify that multiple levels of grouping should be computed at once.

Syntax

```
GROUP BY
{
  column-Name-or-Position ]* |
  ROLLUP ( column-Name-or-Position
            [ , column-Name-or-Position ]* )
}
```

column-Name-or-Position

Must be either the name or position of a column from the current scope of the query; there can be no columns from a query block outside the current scope. For example, if a GROUP BY clause is in a subquery, it cannot refer to columns in the outer query.

Usage Notes

SelectItems in the *SelectExpression* with a GROUP BY clause must contain only aggregates or grouping columns.

Examples

Create our Test Table:

```
CREATE TABLE Test1
(
  TRACK_SEQ VARCHAR(40),
  TRACK_CD VARCHAR(18),
  REC_SEQ_NBR BIGINT,
  INDIV_ID BIGINT,
  BIZ_ID BIGINT,
  ADDR_ID BIGINT,
  HH_ID BIGINT,
  TRIAD_CB_DT DATE
);
```

Populate our Test Table:

```
CREATE TABLE Test1
INSERT INTO Test1 VALUES
  ('1','A',1,1,1,1,1,'2017-07-01'),
  ('1','A',1,1,2,2,2,'2017-07-02'),
  ('3','C',3,1,3,3,3,'2017-07-03'),
  ('1','A',1,2,1,1,1,'2017-07-01'),
  ('1','A',1,2,2,2,2,'2017-07-02'),
  ('3','C',3,2,3,3,3,'2017-07-03');
```

Example: Query Using Column Names:

```
SELECT indiv_id, track_seq, rec_seq_nbr, triad_cb_dt, ROW_NUMBER()
OVER (PARTITION BY indiv_id ORDER BY triad_cb_dt desc,rec_seq_nbr desc) AS ranking
FROM Test1
GROUP BY indiv_id,track_seq,rec_seq_nbr,triad_cb_dt;
INDIV_ID | TRACK_SEQ | REC_SEQ_NBR | TRIAD_CB_&| RANKING
-----
1      | 1          | 1          | 2017-07-01 | 3
1      | 1          | 1          | 2017-07-02 | 2
1      | 3          | 3          | 2017-07-03 | 1
2      | 1          | 1          | 2017-07-01 | 3
2      | 1          | 1          | 2017-07-02 | 2
2      | 3          | 3          | 2017-07-03 | 1
6 rows selected
```

Example: Query Using Column Positions:

```
SELECT indiv_id, track_seq, rec_seq_nbr, triad_cb_dt, ROW_NUMBER()
OVER (PARTITION BY indiv_id ORDER BY triad_cb_dt desc,rec_seq_nbr desc) AS ranking
FROM Test1
GROUP BY 1,2,3,4;
INDIV_ID | TRACK_SEQ | REC_SEQ_NBR | TRIAD_CB_&| RANKING
-----
1      | 1          | 1          | 2017-07-01 | 3
1      | 1          | 1          | 2017-07-02 | 2
1      | 3          | 3          | 2017-07-03 | 1
2      | 1          | 1          | 2017-07-01 | 3
2      | 1          | 1          | 2017-07-02 | 2
2      | 3          | 3          | 2017-07-03 | 1
6 rows selected
```

See Also

» [SELECT expression](#)

HAVING

A HAVING clause restricts the results of a [SelectExpression](#).

The HAVING clause is applied to each group of the grouped table, similarly to how a [WHERE](#) clause is applied to a select list.

If there is no GROUP BY clause, the HAVING clause is applied to the entire result as a single group. The [SELECT](#) expression cannot refer directly to any column that does not have a GROUP BY clause. It can, however, refer to constants, aggregates, and special registers.

Syntax

```
HAVING searchCondition
```

searchCondition

A specialized Boolean expression, as described in the next section.

Using

The *searchCondition*, is a specialized *booleanExpression* that can contain only;

- » grouping columns (see [GROUP BY clause](#))
- » columns that are part of aggregate expressions
- » columns that are part of a subquery

For example, the following query is illegal, because the column SALARY is not a grouping column, it does not appear within an aggregate, and it is not within a subquery:

```
SELECT COUNT(*)
  FROM SAMP.STAFF
 GROUP BY ID
 HAVING SALARY > 15000;
```

Aggregates in the HAVING clause do not need to appear in the SELECT list. If the HAVING clause contains a subquery, the subquery can refer to the outer query block if and only if it refers to a grouping column.

Example

```
-- Find the total number of economy seats taken on a flight,  
-- grouped by airline,  
-- only when the group has at least 2 records.  
SELECT SUM(ECONOMY_SEATS_TAKEN), AIRLINE_FULL  
FROM FLIGHTAVAILABILITY, AIRLINES  
WHERE SUBSTR(FLIGHTAVAILABILITY.FLIGHT_ID, 1, 2) = AIRLINE  
GROUP BY AIRLINE_FULL  
HAVING COUNT(*) > 1;
```

See Also

- » [SELECT expression](#)
- » [GROUP BY clause](#)
- » [WHERE clause](#)

LIMIT n

A `LIMIT n` clause, limits the results of a query to a specified number of records.

Syntax

```
'{` LIMIT {count} '}'
```

NOTE: You must surround the `LIMIT` clause with left and right curly brackets (`{` and `}`).

count

An integer value specifying the maximum number of rows to return from the query.

Examples

```
splice> select * from limittest order by a;
A |B |C |D
```

```
a1 |b1 |c1 |d1
a2 |b2 |c2 |d2
a3 |b3 |c3 |d3
a4 |b4 |c4 |d4
a5 |b5 |c5 |d5
a6 |b6 |c6 |d6
a7 |b7 |c7 |d7
a8 |b8 |c8 |d8
8 rows selected
```

```
splice> select * from limittest order by a {LIMIT 1};
A |B |C |D
```

```
a1 |b1 |c1 |d1
1 row selected
```

```
splice> select * from limittest order by a {LIMIT 3};
A |B |C |D
```

```
a1 |b1 |c1 |d1
a2 |b2 |c2 |d2
a3 |b3 |c3 |d3
3 rows selected
```

```
splice> select * from limittest order by a {LIMIT 10};
A |B |C |D
```

```
a1 |b1 |c1 |d1
a2 |b2 |c2 |d2
a3 |b3 |c3 |d3
a4 |b4 |c4 |d4
a5 |b5 |c5 |d5
a6 |b6 |c6 |d6
a7 |b7 |c7 |d7
a8 |b8 |c8 |d8
8 rows selected
```

See Also

- » [RESULT OFFSET clause](#)
- » [SELECT expression](#)

» **TOP n clause**

ORDER BY

The ORDER BY clause is an optional element of the following:

- » A `SELECT` statement
- » A `SelectExpression`
- » A `VALUES` expression
- » A `ScalarSubquery`
- » A `TableSubquery`

It can also be used in an `CREATE VIEW` statement.

An ORDER BY clause allows you to specify the order in which rows appear in the result set. In subqueries, the ORDER BY clause is meaningless unless it is accompanied by one or both of the `result_offset` and `fetch first` clauses or in conjunction with the `ROW_NUMBER` function, since there is no guarantee that the order is retained in the outer result set. It is permissible to combine ORDER BY on the outer query with ORDER BY in subqueries.

Syntax

```
ORDER BY { column-Name |
            ColumnPosition |
            Expression }
        [ ASC | DESC ]
        [ , column-Name | ColumnPosition | Expression
            [ ASC | DESC ]
            [ NULLS FIRST | NULLS LAST ]
        ]*
    ]*
```

column-Name

A column name, as described in the `SELECT` statement. The column name(s) that you specify in the ORDER BY clause do not need to be the `SELECT` list.

ColumnPosition

An integer that identifies the number of the column in the `SelectItems` in the underlying query of the `SELECT` statement. `ColumnPosition` must be greater than 0 and not greater than the number of columns in the result table. In other words, if you want to order by a column, that column must be specified in the `SELECT` list.

Expression

A sort key expression, such as numeric, string, and datetime expressions. `Expression` can also be a row value expression such as a scalar subquery or case expression.

`ASC`

Specifies that the results should be returned in ascending order. If the order is not specified, `ASC` is the default.

`DESC`

Specifies that the results should be returned in descending order.

NULLS FIRST

Specifies that NULL values should be returned before non-NULL values. This is the default value for descending (DESC) order.

NULLS LAST

Specifies that NULL values should be returned after non-NULL values. This is the default value for ascending (ASC) order.

Using

If `SELECT DISTINCT` is specified or if the `SELECT` statement contains a `GROUP BY` clause, the `ORDER BY` columns must be in the `SELECT` list.

Example using a correlation name

You can sort the result set by a correlation name, if the correlation name is specified in the select list. For example, to return from the `CITIES` database all of the entries in the `CITY_NAME` and `COUNTRY` columns, where the `COUNTRY` column has the correlation name `NATION`, you specify this `SELECT` statement:

```
SELECT CITY_NAME, COUNTRY AS NATION
  FROM CITIES
 ORDER BY NATION;
```

Example using a numeric expression

You can sort the result set by a numeric expression, for example:

```
SELECT name, salary, bonus FROM employee
 ORDER BY salary+bonus;
```

In this example, the `salary` and `bonus` columns are `DECIMAL` data types.

Example using a function

You can sort the result set by invoking a function, for example:

```
SELECT i, len FROM measures
 ORDER BY sin(i);
```

Example of specifying a NULL ordering

You can sort the result set by invoking a function, for example:

```
SELECT * FROM Players  
ORDER BY BirthDate DESC NULLS LAST;
```

See Also

- » [GROUP BY clause](#)
- » [WHERE clause](#)
- » [SELECT expression](#)
- » [VALUES expression](#)
- » [CREATE VIEW statement](#)
- » [INSERT statement](#)
- » [SELECT statement](#)

OVER

The OVER clause is used in window functions to define the window on which the function operates. Window functions are permitted only in the [ORDER BY](#) clause of queries.

For general information about and examples of Window functions in Splice Machine, see the [Using Window Functions](#) topic.

Syntax

```
expression OVER(
    [partitionClause]
    [orderClause]
    [frameClause] );
```

expression

Any value expression that does not itself contain window function calls.

partitionClause

Optional. Specifies how the window function is broken down over groups, in the same way that `GROUP BY` specifies groupings for regular aggregate functions. If you omit this clause, there is one partition that contains all rows.

The syntax for this clause is essentially the same as for the [GROUP BY](#) clause for queries; To recap:

```
PARTITION BY expression [, ...]
```

expression [...]

A list of expressions that define the partitioning.

orderClause

Optional. Controls the ordering. It is important for ranking functions, since it specifies by which variables ranking is performed. It is also needed for cumulative functions. The syntax for this clause is essentially the same as for the [SQL Reference](#). To recap:

```
ORDER BY expression
[ ASC | DESC | USING operator ]
[ NULLS FIRST | NULLS LAST ]
[, ...]
```

NOTE: The default ordering is ascending (ASC). For ascending order, NULL values are returned last unless you specify `NULLS FIRST`; for descending order, NULL values are returned first unless you specify `NULLS LAST`.

frameClause

Optional. Defines which of the rows (which *frame*) that are passed to the window function should be included in the computation. The *frameClause* provides two offsets that determine the start and end of the frame.

The syntax for the frame clause is:

```
[ RANGE | ROWS ] frameStart |
[RANGE | ROWS] BETWEEN frameStart AND frameEnd
```

The syntax for both *frameStart* and *frameEnd* is:

```
UNBOUNDED PRECEDING |
<n> PRECEDING |
CURRENT ROW |
<n> FOLLOWING |
UNBOUNDED FOLLOWING
```

<n>

A non-negative integer value.

Usage Restrictions

Because window functions are only allowed in **HAVING** clauses, you sometimes need to use subqueries with window functions to accomplish what seems like it could be done in a simpler query.

For example, because you cannot use an OVER clause in a WHERE clause, a query like the following is not possible:

```
SELECT *
FROM Batting
WHERE rank() OVER (PARTITION BY "playerID" ORDER BY "G") = 1;
```

And because WHERE and HAVING are computed before the windowing functions, this won't work either:

```
SELECT *, rank() OVER (PARTITION BY "playerID" ORDER BY "G") as rank
FROM Batting
WHERE rank = 1;
```

Instead, you need to use a subquery:

```
SELECT *
FROM (
    SELECT *, rank() OVER (PARTITION BY "playerID" ORDER BY "G") as rank
    FROM Batting
) tmp
WHERE rank = 1;
```

And note that the above subquery will add a rank column to the original columns,

Simple Window Function Examples

The examples in this section are fairly simple because they don't use the frame clause.

```
--- Rank each year within a player by the number of home runs hit by that player
RANK() OVER (PARTITION BY playerID ORDER BY desc(H));

--- Compute the change in number of games played from one year to the next:
G - LAG(G) OVER (PARTITION G playerID ORDER BY yearID);
```

Examples with Frame Clauses

The frame clause can be confusing, given all of the options that it presents. There are three commonly used frame clauses:

Frame Clause Type	Example
<i>Recycled</i>	BETWEEN UNBOUNDED PRECEEDING AND UNBOUNDED FOLLOWING
<i>Cumulative</i>	BETWEEN UNBOUNDED PRECEEDING AND CURRENT ROW
<i>Rolling</i>	BETWEEN 2 PRECEEDING AND 2 FOLLOWING

Here are some examples of window functions using frame clauses:

```
--- Compute the running sum of G for each player:
SUM(G) OVER (PARTITION BY playerID ORDER BY yearID
             BETWEEN UNBOUNDED PRECEEDING AND CURRENT ROW);

--- Compute the career year:
YearID - min(YEARID) OVER (PARTITION BY playerID
                           BETWEEN UNBOUNDED PRECEEDING AND UNBOUNDED FOLLOWING) + 1;

--- Compute a rolling average of games by player:
MEAN(G) OVER (PARTITION BY playerID ORDER BY yearID
              BETWEEN 2 PRECEEDING AND 2 FOLLOWING);
```

See Also

- » [Window and Aggregate functions](#)
- » [SELECT expression](#)

- » [HAVING clause](#)
- » [ORDER BY clause](#)
- » [WHERE clause](#)
- » The [Using Window Functions](#) section in our *Splice Machine Developer's Guide*

RESULT OFFSET and FETCH FIRST

The *result offset clause* provides a way to skip the N first rows in a result set before starting to return any rows.

The *fetch first clause*, which can be combined with the *result offset clause*, limits the number of rows returned in the result set. The *fetch first clause* can sometimes be useful for retrieving only a few rows from an otherwise large result set, usually in combination with an ORDER BY clause. Use of this clause can increase efficiency and make programming simpler.

Syntax

```
OFFSET { integer-literal | ? }
       {ROW | ROWS}
```

integer-literal

An integer value that specifies the number of rows to skip. The default value is 0.

If non-zero, this must be a positive integer value. If you specify a value greater than the number of rows in the underlying result set, no rows are returned.

```
FETCH { FIRST | NEXT }
      [integer-literal | ? ]
      {ROW | ROWS} ONLY
```

integer-literal

An integer value that specifies the maximum number of rows to return in the result set. The default value is 1.

This must be a positive integer value greater than or equal to 1.

Usage

Note that:

- » ROW and ROWS are synonymous
- » FIRST and NEXT are synonymous

Be sure to specify the ORDER BY clause if you expect to retrieve a sorted result set.

Examples

```
-- Fetch the first row of T
SELECT * FROM T FETCH FIRST ROW ONLY;

-- Sort T using column I, then fetch rows 11 through 20
-- of the sorted rows (inclusive)
SELECT * FROM T ORDER BY I
    OFFSET 10 ROWS
    FETCH NEXT 10 ROWS ONLY;

-- Skip the first 100 rows of T
-- If the table has fewer than 101 records,
-- an empty result set is returned
SELECT * FROM T OFFSET 100 ROWS;

-- Use of ORDER BY and FETCH FIRST in a subquery
SELECT DISTINCT A.ORIG_AIRPORT, B.FLIGHT_ID FROM
(SELECT FLIGHT_ID, ORIG_AIRPORT
    FROM FLIGHTS
    ORDER BY ORIG_AIRPORT DESC
    FETCH FIRST 40 ROWS ONLY)
AS A, FLIGHTAVAILABILITY AS B
WHERE A.FLIGHT_ID = B.FLIGHT_ID;

-- JDBC (using a dynamic parameter):
PreparedStatement p =
    con.prepareStatement("SELECT * FROM T
                        ORDER BY I
                        OFFSET ? ROWS");
p.setInt(1, 100);
ResultSet rs = p.executeQuery();
```

See Also

- » [LIMIT n clause](#)
- » [SELECT statement](#)
- » [TOP n clause](#)

TOP n

A TOP clause, also called the TOP n clause, limits the results of a query to the first n result records.

Syntax

```
TOP [number] column-Name]*
```

number

Optional. An integer value that specifies the maximum number of rows to return from the query. If you omit this parameter, the default value of 1 is used.

column-Name

A column name, as described in the [Column Name](#) topic.

You can specify * as the column name to represent all columns.

Examples

```
splice> select * from toptest order by a;
```

```
A |B |C |D
```

```
-----  
a1 |b1 |c1 |d1  
a2 |b2 |c2 |d2  
a3 |b3 |c3 |d3  
a4 |b4 |c4 |d4  
a5 |b5 |c5 |d5  
a6 |b6 |c6 |d6  
a7 |b7 |c7 |d7  
a8 |b8 |c8 |d8  
8 rows selected
```

```
splice> select top * from toptest order by a;
```

```
A |B |C |D
```

```
-----  
a1 |b1 |c1 |d1  
1 row selected
```

```
splice> select top 3 a, b, c from toptest order by a;
```

```
A |B |C
```

```
-----  
a1 |b1 |c1  
a2 |b2 |c2  
a3 |b3 |c3  
3 rows selected
```

```
splice> select top 10 a, b from toptest order by a;
```

```
A |B
```

```
-----  
a1 |b1  
a2 |b2  
a3 |b3  
a4 |b4  
a5 |b5  
a6 |b6  
a7 |b7  
a8 |b8  
8 rows selected
```

```
splice> select top 4 * from toptest order by a offset 1 row;
```

```
A |B |C |D
```

```
-----  
a2 |b2 |c2 |d2  
a3 |b3 |c3 |d3  
a4 |b4 |c4 |d4  
a5 |b5 |c5 |d5  
4 rows selected
```

```
splice> select top 4 * from toptest order by a offset 2 row;
A |B |C |D
```

```
-----  
a3 |b3 |c3 |d3  
a4 |b4 |c4 |d4  
a5 |b5 |c5 |d5  
a6 |b6 |c6 |d6  
4 rows selected
```

```
splice> select top 4 * from toptest order by a offset -1 row ;
ERROR 2201X: Invalid row count for OFFSET, must be >= 0.
```

```
splice> select top 4 * from toptest order by a offset 10 row;
A |B |C |D
```

```
-----  
0 rows selected
```

```
splice> select top -1 * from toptest;
ERROR 2201W: Row count for FIRST/NEXT/TOP must be >= 1 and row count for LIMIT must
be >= 0.
```

See Also

- » [LIMIT n clause](#)
- » [RESULT OFFSET clause](#)
- » [SELECT expression](#)

UNION

The UNION operator combines the result set of two or more similar `SELECT` queries, and returns distinct rows.

Syntax

```
SELECT expression
```

SELECT expression

A SELECT expression that does not include an ORDER BY clause.

If you include an ORDER BY clause, that clause applies to the intersection operation.

DISTINCT

(Optional). Indicates that only distinct (non-duplicate) rows from the queries are included. This is the default.

ALL

(Optional). Indicates that all rows from the queries are included, including duplicates.

Usage

Each SELECT statement in the union must contain the same number of columns, with similar data types, in the same order. Although the number, data types, and order of the fields in the select queries that you combine in a UNION clause must correspond, you can use expressions, such as calculations or subqueries, to make them correspond.

Each UNION keyword combines the SELECT statements that immediately precede and follow it. If you use the ALL keyword with some of the UNION keywords in your query, but not with others, the results will include duplicate rows from the pairs of SELECT statements that are combined by using UNION ALL, but will not include duplicate rows from the SELECT statements that are combined by using UNION without the ALL keyword.

Results

A result set.

Examples

```
CREATE TABLE t1( id INTEGER NOT NULL PRIMARY KEY,
                 i1 INTEGER, i2 INTEGER,
                 c10 char(10), c30 char(30), tm time);

CREATE TABLE t2( id INTEGER NOT NULL PRIMARY KEY,
                 i1 INTEGER, i2 INTEGER,
                 vc20 varchar(20), d double, dt date);

INSERT INTO t1(id,i1,i2,c10,c30) VALUES
(1,1,1,'a','123456789012345678901234567890'),
(2,1,2,'a','bb'),
(3,1,3,'b','bb'),
(4,1,3,'zz','5'),
(5,NULL,NULL,NULL,'1.0'),
(6,NULL,NULL,NULL,'a');

INSERT INTO t2(id,i1,i2,vc20,d) VALUES
(1,1,1,'a',1.0),
(2,1,2,'a',1.1),
(5,NULL,NULL,'12345678901234567890',3),
(100,1,3,'zz',3),
(101,1,2,'bb',NULL),
(102,5,5,' ',NULL),
(103,1,3,' a',NULL),
(104,1,3,'NULL',7.4);
```

```
splice> SELECT id,i1,i2 FROM t1 UNIONSELECT id,i1,i2 FROM t2 ORDER BY id,i1,i2;
```

ID	I1	I2		
2	1	2		
3	1	3		
4	1	3		
5	NULL	NULL		
6	NULL	NULL		
100	1	3		
101	1	2		
102	5	5		
103	1	3		
104	1	3		

11 rows selected

```
splice> SELECT id,i1,i2 FROM t1 UNION ALLSELECT id,i1,i2 FROM t2 ORDER BY id,i1,i2;
ID      | I1      | I2
-----
1       | 1       | 1
1       | 1       | 1
2       | 1       | 2
2       | 1       | 2
3       | 1       | 3
4       | 1       | 3
5       | NULL    | NULL
5       | NULL    | NULL
6       | NULL    | NULL
100     | 1       | 3
101     | 1       | 2
102     | 5       | 5
103     | 1       | 3
104     | 1       | 3

14 rows selected
```

See Also

- » Except clause

USING

The USING clause specifies which columns to test for equality when two tables are joined. It can be used instead of an ON clause in JOIN operations that have an explicit join clause.

Syntax

```
USING ( [ Simple-column-Name ]* )
```

SimpleColumnName

The name of a table column, as described in the Simple Column Name topic.

Using

The columns listed in the USING clause must be present in both of the tables being joined. The USING clause will be transformed to an ON clause that checks for equality between the named columns in the two tables.

When a USING clause is specified, an asterisk (*) in the select list of the query will be expanded to the following list of columns (in this order):

- » All the columns in the USING clause
- » All the columns of the first (left) table that are not specified in the USING clause
- » All the columns of the second (right) table that are not specified in the USING clause

An asterisk qualified by a table name (for example, COUNTRIES.*) will be expanded to every column of that table that is not listed in the USING clause.

If a column in the USING clause is referenced without being qualified by a table name, the column reference points to the column in the first (left) table if the join is a [LEFT OUTER JOIN](#). If it is a [RIGHT OUTER JOIN](#), unqualified references to a column in the USING clause point to the column in the second (right) table.

Examples

The following query performs an inner join between the COUNTRIES table and the CITIES table on the condition that COUNTRIES.COUNTRY is equal to CITIES.COUNTRY:

```
SELECT * FROM COUNTRIES JOIN CITIES
  USING (COUNTRY);
```

The next query is similar to the one above, but it has the additional join condition that COUNTRIES.COUNTRY_ISO_CODE is equal to CITIES.COUNTRY_ISO_CODE:

```
SELECT * FROM COUNTRIES JOIN CITIES  
    USING (COUNTRY, COUNTRY_ISO_CODE);
```

See Also

- » [Join Operations](#)
- » [SELECT statement](#)

WHERE

The WHERE clause is an optional part of an[UPDATE](#) statement.

The WHERE clause lets you select rows based on a Boolean expression. Only rows for which the expression evaluates to TRUE are selected to return or operate upon (delete or update).

Syntax

```
WHERE BooleanExpression
```

BooleanExpression

A Boolean expression. For more information, see the Boolean Expressions topic.

Example

```
-- find the flights where no business-class seats have been booked
SELECT *
  FROM FlightAvailability
 WHERE business_seats_taken IS NULL
   OR business_seats_taken = 0;

-- Join the EMP_ACT and EMPLOYEE tables
-- select all the columns from the EMP_ACT table and
-- add the employee's surname (LASTNAME) from the EMPLOYEE table
-- to each row of the result.
SELECT SAMP.EMP_ACT.* , LASTNAME
  FROM SAMP.EMP_ACT, SAMP.EMPLOYEE
 WHERE EMP_ACT.EMPNO = EMPLOYEE.EMPNO;

-- Determine the employee number and salary of sales representatives
-- along with the average salary and head count of their departments.
-- This query must first create a new-column-name specified in the AS clause
-- which is outside the fullselect (DINFO)
-- in order to get the AVGSALARY and EMPCOUNT columns,
-- as well as the DEPTNO column that is used in the WHERE clause
SELECT THIS_EMP.EMPNO, THIS_EMP.SALARY, DINFO.AVGSALARY, DINFO.EMPCOUNT
  FROM EMPLOYEE THIS_EMP,
       (SELECT OTHERS.WORKDEPT AS DEPTNO,
              AVG(OTHERS.SALARY) AS AVGSALARY,
              COUNT(*) AS EMPCOUNT
     FROM EMPLOYEE OTHERS
    GROUP BY OTHERS.WORKDEPT
   )AS DINFO
 WHERE THIS_EMP.JOB = 'SALESREP'
   AND THIS_EMP.WORKDEPT = DINFO.DEPTNO;
```

See Also

- » [Select expressions](#)
- » [DELETE statement](#)
- » [SELECT statement](#)
- » [UPDATE statement](#)

WITH CLAUSE (Common Table Expression)

You can use Common Table Expressions, also known as the WITH clause, to break down complicated queries into simpler parts by naming and referring to subqueries within queries.

A Common Table Expression (CTE) provides a way of defining a temporary result set whose definition is available only to the query in which the CTE is defined. The result of the CTE is not stored; it exists only for the duration of the query. CTEs are helpful in reducing query complexity and increasing readability. They can be used as substitutions for views in cases where either you don't have permission to create a view or the query would be the only one using the view. CTEs allow you to more easily enable grouping by a column that is derived from a scalar sub select or a function that is non deterministic.

NOTE: The WITH clause is also known as the *subquery factoring clause*.

The handling and syntax of WITH queries are similar to the handling and syntax of views. The WITH clause can be processed as an inline view and shares syntax with CREATE VIEW. The WITH clause can also resolve as a temporary table, which may enhance the efficiency of a query.

Syntax

```
WITH queryName
    AS SELECT Query
```

queryName

An identifier that names the subquery clause.

Restrictions

You cannot currently use a temporary table in a WITH clause. This is being addressed in a future release of Splice Machine.

Examples

If we create the following table:

```
CREATE TABLE BANKS (
    INSTITUTION_ID INTEGER NOT NULL,
    INSTITUTION_NAME VARCHAR(100),
    CITY VARCHAR(100),
    STATE VARCHAR(2),
    TOTAL_ASSETS DECIMAL(19,2),
    NET_INCOME DECIMAL(19,2),
    OFFICES INTEGER,
    PRIMARY KEY(INSTITUTION_ID)
);
```

We can then use a common table expression to improve the readability of a statement that finds the per-city total assets and income for the states with the top net income:

```
WITH state_sales AS (
    SELECT STATE, SUM(NET_INCOME) AS total_sales
    FROM BANKS
    GROUP BY STATE
), top_states AS (
    SELECT STATE
    FROM state_sales
    WHERE total_sales > (SELECT SUM(total_sales)/10 FROM state_sales)
)
SELECT STATE,
    CITY,
    SUM(TOTAL_ASSETS) AS assets,
    SUM(NET_INCOME) AS income
FROM BANKS
WHERE STATE IN (SELECT STATE FROM top_states)
GROUP BY STATE, CITY;
```

See Also

- » [SELECT expression](#)
- » [Query](#)

Expressions

This section contains the reference documentation for the Splice Machine SQL Expressions, in the following topics:

Topic	Description
About Expressions	Overview of expression syntax and rules.
Boolean Expressions	Syntax for and examples of Boolean expressions.
CASE Expression	Syntax for and examples of CASE expressions.
Dynamic Parameters	Description of using dynamic parameters in expressions in prepared statements.
Expression Precedence	Specifies operator precedence in expressions.
NEXT VALUE FOR Expression	Retrieves the next value from a sequence generator.
SELECT Expression	Builds a table value based on filtering and projecting values from other tables.
TABLE Expression	Specifies a table, view, or function in a FROM clause.
VALUES Expression	Constructs a row or a table from other values.



For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](#).

About Expressions

Syntax for many statements and expressions includes the term *Expression*, or a term for a specific kind of expression such as TableSubquery. Expressions are allowed in these specified places within statements.

Some locations allow only a specific type of expression or one with a specific property. If not otherwise specified, an expression is permitted anywhere the word *Expression* appears in the syntax. This includes:

- » ORDER BY clause
- » [SelectExpression](#)
- » UPDATE statement (SET portion)
- » [VALUES Expression](#)
- » WHERE clause

Of course, many other statements include these elements as building blocks, and so allow expressions as part of these elements.

The following tables list all the possible SQL expressions and indicate where the expressions are allowed.

General Expressions

General expressions are expressions that might result in a value of any type. The following table lists the types of general expressions.

Expression Type	Explanation
Column reference	<p>A column-Name that references the value of the column made visible to the expression containing the Column reference.</p> <p>You must qualify the column-Name by the table name or correlation name if it is ambiguous.</p> <p>The qualifier of a column-Name must be the correlation name, if a correlation name is given to a table that is in a SelectExpressions, UPDATE statements, and the WHERE clauses of data manipulation statements.</p>
Constant	Most built-in data types typically have constants associated with them (as shown in the Data types section).
NULL	<p>NULL is an untyped constant representing the unknown value.</p> <p>Allowed in CAST expressions or in INSERT VALUES lists and UPDATE SET clauses. Using it in a CAST expression gives it a specific data type.</p>

Expression Type	Explanation
Dynamic parameter	<p>A dynamic parameter is a parameter to an SQL statement for which the value is not specified when the statement is created. Instead, the statement has a question mark (?) as a placeholder for each dynamic parameter. See Dynamic parameters.</p> <p>Dynamic parameters are permitted only in prepared statements. You must specify values for them before the prepared statement is executed. The values specified must match the types expected.</p> <p>Allowed anywhere in an expression where the data type can be easily deduced. See Dynamic parameters.</p>
CAST expression	<p>Allows you to specify the type of NULL or of a dynamic parameter or convert a value to another type. See CAST function.</p>
Scalar subquery	<p>Subquery that returns a single row with a single column. See ScalarSubquery.</p>
Table subquery	<p>Subquery that returns more than one column and more than one row. See TableSubquery. Allowed as a tableExpression in a FROM clause and with EXISTS, IN, and quantified comparisons.</p>
Conditional expression	<p>A conditional expression chooses an expression to evaluate based on a boolean test. Conditional expressions include the COALESCE function.</p>

Boolean Expressions

Boolean expressions are expressions that result in boolean values. Most general expressions can result in boolean values. Boolean expressions commonly used in a WHERE clause are made of operands operated on by SQL operators.

Numeric Expressions

Numeric expressions are expressions that result in numeric values. Most of the general expressions can result in numeric values. Numeric values have one of the following types:

- » BIGINT
- » DECIMAL
- » DOUBLE PRECISION
- » INTEGER
- » REAL
- » SMALLINT

The following table lists the types of numeric expressions.

Expression Type	Explanation
+, -, *, /, unary + and - expressions	<p>Evaluate the expected math operation on the operands. If both operands are the same type, the result type is not promoted, so the division operator on integers results in an integer that is the truncation of the actual numeric result. When types are mixed, they are promoted as described in the Data types section.</p> <p>Unary + is a noop (i.e., +4 is the same as 4).</p> <p>Unary – is the same as multiplying the value by -1, effectively changing its sign.</p>
AVG	AVG function
SUM	SUM function
LENGTH	LENGTH function.
LOWER	LOWER function.
COUNT	COUNT function, including COUNT(*) .

Character expressions

Character expressions are expressions that result in a CHAR or VARCHAR value. Most general expressions can result in a CHAR or VARCHAR value. The following table lists the types of character expressions.

Expression Type	Explanation
A CHAR or VARCHAR value that uses wildcards.	The wildcards % and _ make a character string a pattern against which the LIKE operator can look for a match.
Concatenation expression	In a concatenation expression, the concatenation operator, , concatenates its right operand to the end of its left operand. Operates on character and bit strings. See Concatenation operator.
Built-in string functions	The built-in string functions act on a String and return a string. See UCASE or UPPER function .

Date and Time Expressions

A date or time expression results in a DATE, TIME, or TIMESTAMP value. Most of the general expressions can result in a date or time value. The following table lists the types of date and time expressions.

Expression Type	Explanation
CURRENT_DATE	Returns the current date. See the CURRENT_DATE function.
CURRENT_TIME	Returns the current time. See the CURRENT_TIME function.
CURRENT_TIMESTAMP	Returns the current timestamp. See the CURRENT_TIMESTAMP function.

See Also

- » [AVG](#) function
- » [CAST](#) function
- » [COUNT](#) function
- » [CURRENT_DATE](#) function
- » [CURRENT_TIME](#) function
- » [CURRENT_TIMESTAMP](#) function
- » [Concatenation](#) operator
- » [LCASE](#) function
- » [LENGTH](#) function
- » [LTRIM](#) function
- » [ORDER BY](#) clause
- » [RTRIM](#) function
- » [SUBSTR](#) function
- » [SUM](#) function
- » [Select](#) expression
- » [TRIM](#) function
- » [UPDATE](#) statement
- » [VALUES](#) expression
- » [WHERE](#) clause

Boolean Expressions

Boolean expressions are allowed in `CONSTRAINT clause` for more information. Boolean expressions in a `WHERE` clause have a highly liberal syntax; see `WHERE clause`, for example.

A Boolean expression can include zero or more Boolean operators.

Syntax

The following table shows the syntax for the Boolean operators

Operator	Syntax
AND, OR, NOT	<pre>{ Expression AND Expression Expression OR Expression NOT Expression }</pre>
Comparisons	<pre>Expression { < = > <= >= <> }</pre>
IS NULL, IS NOT NULL	<pre>Expression IS [NOT] NULL</pre>
LIKE	<pre>CharacterExpression [NOT] LIKE CharacterExpression WithWildCard [ESCAPE 'escapeCharacter']</pre>
BETWEEN	<pre>Expression [NOT] BETWEEN Expression AND Expression</pre>

Operator	Syntax
IN	<pre>{ Expression [NOT] IN TableSubquery Expression [NOT] IN (Expression [, Expression]*) }</pre>
EXISTS	[NOT] EXISTS TableSubquery
Quantified comparison	<pre>Expression ComparisonOperator { ALL ANY SOME } TableSubquery</pre>

Examples

The following example presents examples of the Boolean operators.

Operator	Explanation and Example
AND, OR, NOT	<p>Evaluate any operand(s) that are boolean expressions:</p> <pre>(orig_airport = 'SFO') OR (dest_airport = 'GR U') -- returns true</pre>
Comparisons	<p><, =, >, <=, >=, <> are applicable to all of the built-in types.</p> <pre>DATE('1998-02-26') < DATE('1998-03-01') -- returns true</pre> <p>NOTE: Splice Machine also accepts the != operator, which is not included in the SQL standard.</p>

Operator	Explanation and Example
IS NULL, IS NOT NULL	<p>Test whether the result of an expression is null or not.</p> <pre>WHERE MiddleName IS NULL</pre>
LIKE	<p>Attempts to match a character expression to a character pattern, which is a character string that includes one or more wildcards.</p> <p>% matches any number (zero or more) of characters in the corresponding position in first character expression.</p> <p>_ matches one character in the corresponding position in the character expression.</p> <p>Any other character matches only that character in the corresponding position in the character expression.</p> <pre>city LIKE 'Sant_'</pre> <p>To treat % or _ as constant characters, escape the character with an optional escape character, which you specify with the ESCAPE clause.</p> <pre>SELECT a FROM tabA WHERE a LIKE '%=_' ESCAPE '='</pre> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>NOTE: When LIKE comparisons are used, Splice Machine compares one character at a time for non-metacharacters. This is different than the way Splice Machine processes = comparisons. The comparisons with the = operator compare the entire character string on left side of the = operator with the entire character string on the right side of the = operator.</p> </div>

Operator	Explanation and Example
BETWEEN	<p>Tests whether the first operand is between the second and third operands. The second operand must be less than the third operand. Applicable only to types to which <code><=</code> and <code>>=</code> can be applied.</p> <pre>WHERE booking_date BETWEEN DATE('1998-02-26') AND DATE('1998-03-01')</pre> <p>NOTE: Using the BETWEEN operator is logically equivalent to specifying that you want to select values that are greater than or equal to the first operand and less than or equal to the second operand: <code>col between X and Y</code> is equivalent to <code>col >= X and col <= Y</code>. Which means that the result set will be empty if your second operand is less than your first.</p>
IN	<p>Operates on table subquery or list of values. Returns TRUE if the left expression's value is in the result of the table subquery or in the list of values. Table subquery can return multiple rows but must return a single column.</p> <pre>WHERE booking_date NOT IN (SELECT booking_date FROM HotelBookings WHERE rooms_available = 0)</pre>
EXISTS	<p>Operates on a table subquery. Returns TRUE if the table subquery returns any rows, and FALSE if it returns no rows. A table subquery can return multiple columns and rows.</p> <pre>WHERE EXISTS (SELECT * FROM Flights WHERE dest_airport = 'SFO' AND orig_airport = 'GRU')</pre>

Operator	Explanation and Example
Quantified comparison	<p>A quantified comparison is a comparison operator (<code><</code>, <code>=</code>, <code>></code>, <code><=</code>, <code>>=</code>, <code><></code>) with <code>ALL</code> or <code>ANY</code> or <code>SOME</code> applied.</p> <p>Operates on table subqueries, which can return multiple rows but must return a single column.</p> <p>If <code>ALL</code> is used, the comparison must be true for all values returned by the table subquery. If <code>ANY</code> or <code>SOME</code> is used, the comparison must be true for at least one value of the table subquery. <code>ANY</code> and <code>SOME</code> are equivalent.</p> <pre>WHERE normal_rate < ALL (SELECT budget/550 FROM Groups)</pre>

See Also

- » [CONSTRAINT clause](#)
- » [WHERE clause](#)

CASE Expression

The CASE expression can be used for conditional expressions in Splice Machine.

Syntax

You can place a CASE expression anywhere an expression is allowed. It chooses an expression to evaluate based on a boolean test.

```
CASE
  WHEN booleanExpression THEN thenExpression
  [ WHEN booleanExpression
    THEN thenExpression ]...
  ELSE elseExpression
END
```

thenExpression and elseExpression

Both are both that must be type-compatible. For built-in types, this means that the types must be the same or a built-in broadening conversion must exist between the types.

Example

```
-- returns 3
CASE WHEN 1=1 THEN 3 ELSE 4 END;

-- returns 7
CASE
  WHEN 1 = 2 THEN 3
  WHEN 4 = 5 THEN 6
  ELSE 7
END;
```

Dynamic Parameters

You can prepare statements that are allowed to have parameters for which the value is not specified when the statement is repared using *PreparedStatement* methods in the JDBC API. These parameters are called dynamic parameters and are represented by a ?.

The JDBC API documents refer to dynamic parameters as IN, INOUT, or OUT parameters. In SQL, they are always IN parameters.

You must specify values for dynamic parameters before executing the statement, and the types of the specified values must match the expected types.

Example

```
PreparedStatement ps2 = conn.prepareStatement(
    "UPDATE HotelAvailability SET rooms_available = " +
    "(rooms_available - ?) WHERE hotel_id = ? " +
    "AND booking_date BETWEEN ? AND ?");

    -- this sample code sets the values of dynamic parameters
    -- to be the values of program variables
ps2.setInt(1, numberRooms);
ps2.setInt(2, theHotel.hotelId);
ps2.setDate(3, arrival);
ps2.setDate(4, departure);
updateCount = ps2.executeUpdate();
```

Where Dynamic Parameters are Allowed

You can use dynamic parameters anywhere in an expression where their data type can be easily deduced.

- » Use as the first operand of BETWEEN is allowed if one of the second and third operands is not also a dynamic parameter. The type of the first operand is assumed to be the type of the non-dynamic parameter, or the union result of their types if both are not dynamic parameters.

```
WHERE ? BETWEEN DATE('1996-01-01') AND ?
    -- types assumed to be DATE
```

- » Use as the second or third operand of BETWEEN is allowed. Type is assumed to be the type of the left operand.

```
WHERE DATE('1996-01-01') BETWEEN ? AND ?
    -- types assumed to be DATE
```

- » Use as the left operand of an IN list is allowed if at least one item in the list is not itself a dynamic parameter. Type for the left operand is assumed to be the union result of the types of the non-dynamic parameters in the list.

```
WHERE ? NOT IN (?, ?, 'Santiago')
-- types assumed to be CHAR
```

- » Use in the values list in an `IN` predicate is allowed if the first operand is not a dynamic parameter or its type was determined in the previous rule. Type of the dynamic parameters appearing in the values list is assumed to be the type of the left operand.

```
WHERE FloatColumn IN (?, ?, ?)
-- types assumed to be FLOAT
```

- » For the binary operators `+`, `-`, `*`, `/`, `AND`, `OR`, `<`, `>`, `=`, `<>`, `<=`, and `=`` use of a dynamic parameter as one operand but not both is permitted. Its type is taken from the other side.

```
WHERE ? < CURRENT_TIMESTAMP
-- type assumed to be a TIMESTAMP
```

- » Use in a `CAST` is always permitted. This gives the dynamic parameter a type.

```
CALL valueOf(CAST (? AS VARCHAR(10)))
```

- » Use on either or both sides of `LIKE` operator is permitted. When used on the left, the type of the dynamic parameter is set to the type of the right operand, but with the maximum allowed length for the type. When used on the right, the type is assumed to be of the same length and type as the left operand. (`LIKE` is permitted on `CHAR` and `VARCHAR` types; see Concatenation operator for more information.)

```
WHERE ? LIKE 'Santi%'
-- type assumed to be CHAR with a length of
-- java.lang.Integer.MAX_VALUE
```

- » In a conditional expression, which uses a `?`, use of a dynamic parameter (which is also represented as a `?`) is allowed. The type of a dynamic parameter as the first operand is assumed to be boolean. Only one of the second and third operands can be a dynamic parameter, and its type will be assumed to be the same as that of the other (that is, the third and second operand, respectively).

```
SELECT c1 IS NULL ? : c1
-- allows you to specify a "default" value at execution time
-- dynamic parameter assumed to be the type of c1
-- you cannot have dynamic parameters on both sides
-- of the :
```

- » A dynamic parameter is allowed as an item in the values list or select list of an `INSERT` statement. The type of the dynamic parameter is assumed to be the type of the target column.

```
INSERT INTO t VALUES (?)
-- dynamic parameter assumed to be the type
-- of the only column in table t
INSERT INTO t SELECT ?
FROM t2
-- not allowed
```

- » A `?` parameter in a comparison with a subquery takes its type from the expression being selected by the subquery. For

example:

```
SELECT *
FROM tab1
WHERE ? = (SELECT x FROM tab2)
SELECT *
FROM tab1
WHERE ? = ANY (SELECT x FROM tab2)
-- In both cases, the type of the dynamic parameter is
-- assumed to be the same as the type of tab2.x.
```

- » A dynamic parameter is allowed as the value in an UPDATE statement. The type of the dynamic parameter is assumed to be the type of the column in the target table.

```
UPDATE t2 SET c2 =?
-- type is assumed to be type of c2
```

- » Dynamic parameters are allowed as the operand of the unary operators – or +. For example:

```
CREATE TABLE t1 (c11 INT, c12 SMALLINT, c13 DOUBLE, c14 CHAR(3))
SELECT * FROM t1 WHERE c11 BETWEEN -? AND +?
-- The type of both of the unary operators is INT
-- based on the context in which they are used (that is,
-- because c11 is INT, the unary parameters also get the
-- type INT.
```

- » LENGTH allow a dynamic parameter. The type is assumed to be a maximum length VARCHAR type.

```
SELECT LENGTH(?)
```

- » Qualified comparisons.

```
? = SOME (SELECT 1 FROM t)
-- is valid. Dynamic parameter assumed to be INTEGER type

1 = SOME (SELECT ? FROM t)
-- is valid. Dynamic parameter assumed to be INTEGER type.
```

- » A dynamic parameter is allowed as the left operand of an IS expression and is assumed to be a Boolean.

Expression Precedence

The precedence of operations from highest to lowest is:

- » $()$, $?$, Constant (including sign), `NULL`, `ColumnReference`, `ScalarSubquery`, `CAST`
- » `LENGTH`, `CURRENT_DATE`, `CURRENT_TIME`, `CURRENT_TIMESTAMP`, and other built-ins
- » unary $+$ and $-$
- » $*$, $/$, $\|$ (concatenation)
- » binary $+$ and $-$
- » comparisons, quantified comparisons, `EXISTS`, `IN`, `IS NULL`, `LIKE`, `BETWEEN`, `IS`
- » `NOT`
- » `AND`
- » `OR`

You can explicitly specify precedence by placing expressions within parentheses. An expression within parentheses is evaluated before any operations outside the parentheses are applied to it.

Example

```
(3+4)*9  
(age < 16 OR age > 65) AND employed = TRUE
```

NEXT VALUE FOR Expression

The `NEXT VALUE FOR` expression retrieves the next value from a sequence generator that was created with a [CREATE SEQUENCE](#) statement.

Syntax

```
NEXT VALUE FOR sequenceName
```

sequenceName

A sequence name is an identifier that can optionally be qualified by a schema name:

[`SQLIdentifier`]

If *schemaName* is not provided, the current schema is the default schema. If a qualified sequence name is specified, the schema name cannot begin with the `SYS.` prefix.

Usage

If this is the first use of the sequence generator, the generator returns its `START` value. Otherwise, the `INCREMENT` value is added to the previous value returned by the sequence generator. The data type of the value is the *dataType* specified for the sequence generator.

If the sequence generator wraps around, then one of the following happens:

- » If the sequence generator was created using the `CYCLE` keyword, the sequence generator is reset to its `START` value.
- » If the sequence generator was created with the default `NO CYCLE` behavior, Splice Machine throws an exception.

In order to retrieve the next value of a sequence generator, you or your session's current role must have `USAGE` privilege on the generator.

A `NEXT VALUE FOR` expression may occur in the following places:

- » `SELECT` statement: As part of the expression defining a returned column in a `SELECT` list
- » `VALUES` expression: As part of the expression defining a column in a row constructor (`VALUES` expression)
- » `UPDATE` statement: As part of the expression defining the new value to which a column is being set

The next value of a sequence generator is not affected by whether the user commits or rolls back a transaction which invoked the sequence generator.

Restrictions

Only one `NEXT VALUE FOR` expression is allowed per sequence per statement.

The `NEXT VALUE FOR` expression is not allowed in any statement which has a `DISTINCT` or `ORDER BY` expression.

A `NEXT VALUE` expression **may not appear** in any of these situations:

- » `CASE` expression
- » `WHERE` clause
- » `ORDER BY` clause
- » Aggregate expression
- » Window functions
- » `ROW_NUMBER` function
- » `DISTINCT` select list

Examples

```
VALUES (NEXT VALUE FOR order_id);

INSERT INTO re_order_table
  SELECT NEXT VALUE FOR order_id, order_date, quantity
    FROM orders
   WHERE back_order = 1;

UPDATE orders
  SET oid = NEXT VALUE FOR order_id
 WHERE expired = 1;
```

See Also

- » `CREATE SEQUENCE` function
- » `SELECT` statement
- » `VALUES` expression
- » `UPDATE` statement

SELECT Expression

A *SelectExpression* is the basic SELECT–FROM–WHERE construct used to build a table value based on filtering and projecting values from other tables.

Syntax

```
SELECT [ DISTINCT | ALL ] SelectItem [ , SelectItem ]*
  FROM clause
  [ WHERE clause ]
  [ GROUP BY clause ]
  [ HAVING clause ]
  [ ORDER BY clause ]
  [ result offset clause ]
  [ fetch first clause ]
```

SELECT clause

The *SELECT* clause contains a list of expressions and an optional quantifier that is applied to the results of the *WHERE* clause.

If *DISTINCT* is specified, only one copy of any row value is included in the result. Nulls are considered duplicates of one another for the purposes of *DISTINCT*.

If no quantifier, or *ALL*, is specified, no rows are removed from the result in applying the *SELECT* clause. This is the default behavior.

SelectItem:

```
{
  * |
  { <a href="correlation-Name" >.* |
    Expression [AS Simple-column-Name] }
```

A *SelectItem* projects one or more result column values for a table result being constructed in a *SelectExpression*.

For queries that do not select a specific column from the tables involved in the *SelectExpression* (for example, queries that use *COUNT(*)*), the user must have at least one column-level *SELECT* privilege or table-level *SELECT* privilege. See *GRANT* statement for more information.

FROM clause

The result of the *FROM* clause is the cross product of the *FROM* items.

WHERE clause

The *WHERE* clause can further qualify the result of the *FROM* clause.

GROUP BY clause

The `GROUP BY` clause groups rows in the result into subsets that have matching values for one or more columns.

`GROUP BY` clauses are typically used with aggregates. If there is a `GROUP BY` clause, the `SELECT` clause must contain *only* aggregates or grouping columns. If you want to include a non-grouped column in the `SELECT` clause, include the column in an aggregate expression. For example, this query computes the average salary of each team in a baseball league:

```
splice> SELECT COUNT(*) AS PlayerCount, Team, AVG(Salary) AS AverageSalary
      FROM Players JOIN Salaries ON Players.ID=Salaries.ID
      GROUP BY Team
      ORDER BY AverageSalary;
```

If there is no `GROUP BY` clause, but a `SelectItem` contains an aggregate not in a subquery, the query is implicitly grouped. The entire table is the single group.

`HAVING` clause

The `HAVING` clause can further qualify the result of the `FROM` clause. This clause restricts a grouped table, specifying a search condition (much like a `WHERE` clause) that can refer only to grouping columns or aggregates from the current scope.

The `HAVING` clause is applied to each group of the grouped table. If the `HAVING` clause evaluates to `TRUE`, the row is retained for further processing; if it evaluates to `FALSE` or `NULL`, the row is discarded. If there is a `HAVING` clause but no `GROUP BY`, the table is implicitly grouped into one group for the entire table.

`ORDER BY` clause

The `ORDER BY` clause allows you to specify the order in which rows appear in the result set. In subqueries, the `ORDER BY` clause is meaningless unless it is accompanied by one or both of the `result offset` and `fetch first` clauses.

`result offset` and `fetch first` clauses

The `fetch first` clause, which can be combined with the `result offset` clause, limits the number of rows returned in the result set.

Usage

The result of a `SelectExpression` is always a table.

Splice Machine processes the clauses in a `Select` expression in the following order:

- » `FROM` clause
- » `WHERE` clause
- » `GROUP BY` (or implicit `GROUP BY`)
- » `HAVING` clause
- » `ORDER BY` clause
- » `Result offset` clause
- » `Fetch first` clause

» SELECT clause

When a query does not have a `FROM` clause (when you are constructing a value, not getting data out of a table), use a `VALUES` expression, not a `SelectExpression`. For example:

```
VALUES CURRENT_TIMESTAMP;
```

The * wildcard

The wildcard character (`**`) expands to all columns in the tables in the associated `FROM` clause.

- » correlation-Name identifiers expand to all columns in the identified table. That table must be listed in the associated `FROM` clause.

Naming columns

You can name a `SelectItem` column using the `AS` clause.

If a column of a `SelectItem` is not a simple `ColumnReference` expression or named with an `AS` clause, it is given a generated unique name.

These column names are useful in several cases:

- » They are made available on the JDBC `ResultSetMetaData`.
- » They are used as the names of the columns in the resulting table when the `SelectExpression` is used as a table subquery in a `FROM` clause.
- » They are used in the `ORDER BY` clause as the column names available for sorting.

Examples

This example shows using a `SELECT` with `WHERE` and `ORDER BY` clauses; it selects the name, team, and birth date of all players born in 1985 and 1989:

```
splice> SELECT DisplayName, Team, BirthDate
   FROM Players
  WHERE YEAR(BirthDate) IN (1985, 1989)
  ORDER BY BirthDate;
DISPLAYNAME          | TEAM      | BIRTHDATE
-----
Jeremy Johnson       | Cards     | 1985-03-15
Gary Kosovo          | Giants    | 1985-06-12
Michael Hillson     | Cards     | 1985-11-07
Mitch Canepa         | Cards     | 1985-11-26
Edward Erdman       | Cards     | 1985-12-21
Jeremy Packman      | Giants    | 1989-01-01
Nathan Nickels      | Giants    | 1989-05-04
Ken Straiter        | Cards     | 1989-07-20
Marcus Bamburger   | Giants    | 1989-08-01
George Goomba        | Cards     | 1989-08-08
Jack Hellman         | Cards     | 1989-08-09
Elliot Andrews       | Giants    | 1989-08-21
Henry Socomy         | Giants    | 1989-11-17

```

13 rows selected

This example shows using correlation names for the tables:

```
splice> SELECT CONSTRAINTNAME, COLUMNNAME
   FROM SYS.SYSTABLES t, SYS.SYSCOLUMNS col,
SYS.SYSCONSTRAINTS cons, SYS.SYSCHECKS checks
  WHERE t.TABLENAME = 'FLIGHTS'
    AND t.TABLEID = col.REFERENCEID
    AND t.TABLEID = cons.TABLEID
    AND cons.CONSTRAINTID = checks.CONSTRAINTID
  ORDER BY CONSTRAINTNAME;
```

This example shows using the DISTINCT clause:

```
SELECT DISTINCT SALARY   FROM Salaries;
```

This example shows how to rename an expression. We use the name BOSS as the maximum department salary for all departments whose maximum salary is less than the average salary in all other departments:

```
SELECT WORKDEPT AS DPT, MAX(SALARY) AS BOSS
   FROM EMPLOYEE EMP_COR
  GROUP BY WORKDEPT
 HAVING MAX(SALARY) <      (SELECT AVG(SALARY)
   FROM EMPLOYEE
  WHERE NOT WORKDEPT = EMP_COR.WORKDEPT)
  ORDER BY BOSS;
```

See Also

- » [FROM clause](#)
- » [GROUP BY clause](#)
- » [HAVING clause](#)
- » [ORDER BY clause](#)
- » [WHERE clause](#)

TABLE Expression

A *TableExpression* specifies a table, view, or function in a `FROM` clause. It is the source from which a [*TableExpression*](#) selects a result.

Syntax

```
{  
    JOIN operations  
}
```

Usage

A correlation name can be applied to a table in a *TableExpression* so that its columns can be qualified with that name.

- » If you do not supply a correlation name, the table name qualifies the column name.
- » When you give a table a correlation name, you cannot use the table name to qualify columns.
- » You must use the correlation name when qualifying column names.
- » No two items in the `FROM` clause can have the same correlation name, and no correlation name can be the same as an unqualified table name specified in that `FROM` clause.

In addition, you can give the columns of the table new names in the `AS` clause. Some situations in which this is useful:

- » When a `TableSubquery`, since there is no other way to name the columns of a `VALUES` expression.
- » When column names would otherwise be the same as those of columns in other tables; renaming them means you don't have to qualify them.

The Query in a *TableSubquery*.

Example

```
-- SELECT from a JOIN expression  
SELECT E.EMPNO, E.LASTNAME, M.EMPNO, M.LASTNAME  
  FROM EMPLOYEE E LEFT OUTER JOIN  
        DEPARTMENT INNER JOIN EMPLOYEE M  
      ON MGRNO = M.EMPNO  
      ON E.WORKDEPT = DEPTNO;
```

TableViewOrFunctionExpression

```
{
  { view-Name }
  [ CorrelationClause ] |
  { TableSubquery | TableFunctionInvocation }
  CorrelationClause
}
```

where *CorrelationClause* is

```
[ AS ]
correlation-Name
[ ( Simple-column-Name * ) ]
```

TableFunctionExpression

```
{
  TABLE function-name( [ [ function-arg ] [, function-arg ]* ] )
```

Note that when you invoke a table function, you must bind it to a correlation name. For example:

```
splice> SELECT s.* FROM TABLE( externalEmployees( 42 ) ) s;
```

See Also

- » [FROM clause](#)
- » [JOIN operations](#)
- » [SELECT statement](#)
- » [VALUES expression](#)

VALUES Expression

The VALUES expression allows construction of a row or a table from other values.

Syntax

```
{
  VALUES ( Value {, Value }* )
  [ , ( Value {, Value }* ) ]* |
  VALUES Value [ , Value ]*
}
[ ORDER BY clause ]
[ result offset clause ]
[ fetch first clause ]
```

Value

Expression | DEFAULT

The first form constructs multi-column rows. The second form constructs single-column rows, each expression being the value of the column of the row.

The `DEFAULT` keyword is allowed only if the `VALUES` expression is in an `INSERT` statement. Specifying `DEFAULT` for a column inserts the column's default value into the column. Another way to insert the default value into the column is to omit the column from the column list and only insert values into other columns in the table.

ORDER BY clause

The `ORDER BY` clause allows you to specify the order in which rows appear in the result set.

result offset and fetch first clauses

The `fetch first` clause, which can be combined with the `result offset` clause, limits the number of rows returned in the result set.

Usage

A `VALUES` expression can be used in all the places where a query can, and thus can be used in any of the following ways:

- » As a statement that returns a `ResultSet`
- » Within expressions and statements wherever subqueries are permitted
- » As the source of values for an `INSERT` statement (in an `INSERT` statement, you normally use a `VALUES` expression when you do not use a `SelectExpression`)

You can use a `VALUES` expression to generate new data values with a query that selects from a `VALUES` clause; for example:

```
SELECT R1,R2  
  FROM (VALUES('GROUP 1','GROUP 2')) AS MYTBL(R1,R2);
```

A VALUES expression that is used in an INSERT statement cannot use an ORDER BY clause. However, if the VALUES expression does not contain the DEFAULT keyword, the VALUES clause can be put in a subquery and ordered, as in the following statement:

```
INSERT INTO t SELECT * FROM (VALUES 'a','c','b') t ORDER BY 1;
```

Examples

```
-- 3 rows of 1 column
splice> VALUES (1),(2),(3);

-- 3 rows of 1 column
splice> VALUES 1, 2, 3;

-- 1 row of 3 columns
splice> VALUES (1, 2, 3);

-- 3 rows of 2 columns
splice> VALUES (1,21),(2,22),(3,23);

-- using ORDER BY and FETCH FIRST
splice> VALUES (3,21),(1,22),(2,23) ORDER BY 1 FETCH FIRST 2 ROWS ONLY;

-- using ORDER BY and OFFSET
splice> VALUES (3,21),(1,22),(2,23) ORDER BY 1 OFFSET 1 ROW;

-- constructing a derived table
splice> VALUES ('orange', 'orange'), ('apple', 'red'), ('banana', 'yellow');

-- Insert two new departments using one statement into the DEPARTMENT table,
-- but do not assign a manager to the new department.
splice> INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT)
VALUES ('B11', 'PURCHASING', 'B01'),
('E41', 'DATABASE ADMINISTRATION', 'E01');

-- insert a row with a DEFAULT value for the MAJPROJ column
splice> INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTDATE, MAJPROJ)
VALUES ('PL2101', 'ENSURE COMPAT PLAN', 'B01', '000020', CURRENT_DATE, DEFAULT);

-- using a built-in function
splice> VALUES CURRENT_DATE;

-- getting the value of an arbitrary expression
splice> VALUES (3*29, 26.0E0/3);

-- getting a value returned by a built-in function
splice> values char(1);
```

See Also

- » [FROM clause](#)
- » [ORDER BY clause](#)
- » [INSERT statement](#)

Join Operations

This section contains the reference documentation for the Splice Machine SQL Join Operations, in the following topics:

Topic	Description
About Join Operations	Overview of joins.
CROSS JOIN	Produces the Cartesian product of two tables: it produces rows that combine each row from the first table with each row from the second table.
INNER JOIN	Selects all rows from both tables as long as there is a match between the columns in both tables.
LEFT OUTER JOIN	Returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.
NATURAL JOIN	Creates an implicit join clause for you based on the common columns (those with the same name in both tables) in the two tables being joined.
RIGHT OUTER JOIN	Returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.



For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](#).

About Join Operations

The JOIN operations, which are among the possible *FROM clause*, perform joins between two tables.

Syntax

JOIN Operation

The following table describes the JOIN operations:

Join Operation	Description
INNER JOIN	Specifies a join between two tables with an explicit join clause.
LEFT OUTER JOIN	Specifies a join between two tables with an explicit join clause, preserving unmatched rows from the first table.
RIGHT OUTER JOIN	Specifies a join between two tables with an explicit join clause, preserving unmatched rows from the second table.
CROSS JOIN	Specifies a join that produces the Cartesian product of two tables. It has no explicit join clause.
NATURAL JOIN	Specifies an inner or outer join between two tables. It has no explicit join clause. Instead, one is created implicitly using the common columns from the two tables. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> NOTE: Splice Machine does not currently support NATURAL SELF JOIN operations. </div>

In all cases, you can specify additional restrictions on one or both of the tables being joined in outer join clauses or in the *WHERE clause*.

Usage

Note that you can also perform a join between two tables using an explicit equality test in a *WHERE clause*, such as:

```
WHERE t1.col1 = t2.col2.
```

See Also

- » [FROM clause](#)
- » [JOIN operations](#)
- » [TABLE expressions](#)
- » [WHERE clause](#)

CROSS JOIN

A CROSS JOIN is a [JOIN](#) operation that produces the Cartesian product of two tables. Unlike other JOIN operators, it does not let you specify a join clause. You may, however, specify a WHERE clause in the SELECT statement.

Syntax

```
TableExpression CROSS JOIN ( TableExpression )
```

Examples

The following SELECT statements are equivalent:

```
splice> SELECT * FROM CITIES CROSS JOIN FLIGHTS;
splice> SELECT * FROM CITIES, FLIGHTS;
```

The following SELECT statements are equivalent:

```
splice> SELECT * FROM CITIES CROSS JOIN FLIGHTS
      WHERE CITIES.AIRPORT = FLIGHTS.ORIG_AIRPORT;
splice> SELECT * FROM CITIES INNER JOIN FLIGHTS
      ON CITIES.AIRPORT = FLIGHTS.ORIG_AIRPORT;
```

The following example is more complex. The ON clause in this example is associated with the LEFT OUTER JOIN operation. Note that you can use parentheses around a JOIN operation.

```
splice> SELECT * FROM CITIES LEFT OUTER JOIN
      (FLIGHTS CROSS JOIN COUNTRIES)
      ON CITIES.AIRPORT = FLIGHTS.ORIG_AIRPORT
      WHERE COUNTRIES.COUNTRY_ISO_CODE = 'US';
```

A CROSS JOIN operation can be replaced with an INNER JOIN where the join clause always evaluates to true (for example, 1=1). It can also be replaced with a sub-query. So equivalent queries would be:

```
splice> SELECT * FROM CITIES LEFT OUTER JOIN
    FLIGHTS INNER JOIN COUNTRIES ON 1=1
    ON CITIES.AIRPORT = FLIGHTS.ORIG_AIRPORT
    WHERE COUNTRIES.COUNTRY_ISO_CODE = 'US';

splice> SELECT * FROM CITIES LEFT OUTER JOIN
    (SELECT * FROM FLIGHTS, COUNTRIES) S
    ON CITIES.AIRPORT = S.ORIG_AIRPORT
    WHERE S.COUNTRY_ISO_CODE = 'US';
```

See Also

- » [JOIN operations](#)
- » [USING clause](#)

INNER JOIN

An INNER JOIN is a [JOIN](#) operation that allows you to specify an explicit join clause.

Syntax

```
TableExpression  
{ ON booleanExpression | USING clause }
```

You can specify the join clause by specifying ON with a boolean expression.

The scope of expressions in the ON clause includes the current tables and any tables in outer query blocks to the current SELECT. In the following example, the ON clause refers to the current tables:

```
SELECT *  
FROM SAMP.EMPLOYEE INNER JOIN SAMP.STAFF  
ON EMPLOYEE.SALARY < STAFF.SALARY;
```

The ON clause can reference tables not being joined and does not have to reference either of the tables being joined (though typically it does).

Examples

```

-- Join the EMP_ACT and EMPLOYEE tables
-- select all the columns from the EMP_ACT table and
-- add the employee's surname (LASTNAME) from the EMPLOYEE table
-- to each row of the result
splice> SELECT SAMP.EMP_ACT.* , LASTNAME
  FROM SAMP.EMP_ACT JOIN SAMP.EMPLOYEE
  ON EMP_ACT.EMPNO = EMPLOYEE.EMPNO;

-- Join the EMPLOYEE and DEPARTMENT tables,
-- select the employee number (EMPNO),
-- employee surname (LASTNAME),
-- department number (WORKDEPT in the EMPLOYEE table and DEPTNO in the
-- DEPARTMENT table)
-- and department name (DEPTNAME)
-- of all employees who were born (BIRTHDATE) earlier than 1930.
splice> SELECT EMPNO, LASTNAME, WORKDEPT, DEPTNAME
  FROM SAMP.EMPLOYEE JOIN SAMP.DEPARTMENT
  ON WORKDEPT = DEPTNO
  AND YEAR(BIRTHDATE) < 1930;

-- Another example of "generating" new data values,
-- using a query which selects from a VALUES clause (which is an
-- alternate form of a fullselect).
-- This query shows how a table can be derived called "X"
-- having 2 columns "R1" and "R2" and 1 row of data
splice> SELECT *
  FROM (VALUES (3, 4), (1, 5), (2, 6))
  AS VALUESTABLE1(C1, C2)
  JOIN (VALUES (3, 2), (1, 2),
(0, 3)) AS VALUESTABLE2(c1, c2)
  ON VALUESTABLE1.c1 = VALUESTABLE2.c1;
  -- This results in:
  -- C1      |C2      |C1      |2
  -- -----|-----|-----|-----
  -- 3      | 4      | 3      | 2
  -- 1      | 5      | 1      | 2

-- List every department with the employee number and
-- last name of the manager
splice> SELECT DEPTNO, DEPTNAME, EMPNO, LASTNAME
  FROM DEPARTMENT INNER JOIN EMPLOYEE
  ON MGRNO = EMPNO;

-- List every employee number and last name
-- with the employee number and last name of their manager
splice> SELECT E.EMPNO, E.LASTNAME, M.EMPNO, M.LASTNAME
  FROM EMPLOYEE E INNER JOIN
  DEPARTMENT INNER JOIN EMPLOYEE M
  ON MGRNO = M.EMPNO
  ON E.WORKDEPT = DEPTNO;

```

See Also

- » JOIN operations
- » USING clause

LEFT OUTER JOIN

A `LEFT OUTER JOIN` is one of the [JOIN](#) operations that allow you to specify a join clause. It preserves the unmatched rows from the first (left) table, joining them with a `NONE` row in the shape of the second (right) table.

Syntax

```
TableExpression
{
    ON booleanExpression |
    USING clause
}
```

The scope of expressions in either the `ON` clause includes the current tables and any tables in query blocks outer to the current `SELECT`. The `ON` clause can reference tables not being joined and does not have to reference either of the tables being joined (though typically it does).

Example 1

```
-- match cities to countries in Asia
splice> SELECT CITIES.COUNTRY, CITIES.CITY_NAME, REGION
      FROM Countries
      LEFT OUTER JOIN Cities
      ON CITIES.COUNTRY_ISO_CODE = COUNTRIES.COUNTRY_ISO_CODE
      WHERE REGION = 'Asia';

-- use the synonymous syntax, LEFT JOIN, to achieve exactly
-- the same results as in the example above
splice> SELECT COUNTRIES.COUNTRY, CITIES.CITY_NAME, REGION
      FROM COUNTRIES
      LEFT JOIN CITIES
      ON CITIES.COUNTRY_ISO_CODE = COUNTRIES.COUNTRY_ISO_CODE
      WHERE REGION = 'Asia';
```

Example 2

```
-- Join the EMPLOYEE and DEPARTMENT tables,
-- select the employee number (EMPNO),
-- employee surname (LASTNAME),
-- department number (WORKDEPT in the EMPLOYEE table
-- and DEPTNO in the DEPARTMENT table)
-- and department name (DEPTNAME)
-- of all employees who born (BIRTHDATE) earlier than 1930
splice> SELECT EMPNO, LASTNAME, WORKDEPT, DEPTNAME
  FROM SAMP.EMPLOYEE LEFT OUTER JOIN SAMP.DEPARTMENT
  ON WORKDEPT = DEPTNO
  AND YEAR(BIRTHDATE) < 1930;

-- List every department with the employee number and
-- last name of the manager,
-- including departments without a manager
splice> SELECT DEPTNO, DEPTNAME, EMPNO, LASTNAME
  FROM DEPARTMENT LEFT OUTER JOIN EMPLOYEE
  ON MGRNO = EMPNO;
```

See Also

- » [JOIN operations](#)
- » [TABLE expression](#)
- » [USING clause](#)

NATURAL JOIN

A NATURAL JOIN is a [JOIN](#) operation that creates an implicit join clause for you based on the common columns in the two tables being joined. Common columns are columns that have the same name in both tables.

Syntax

```
TableExpression NATURAL
[ { LEFT | RIGHT }
  [ OUTER ] | INNER ] JOIN
{ TableViewOrFunctionExpression | ( TableExpression ) }
```

Usage

A NATURAL JOIN can be an INNER join, a LEFT OUTER join, or a RIGHT OUTER join. The default is INNER join.

If the SELECT statement in which the NATURAL JOIN operation appears has an asterisk (*) in the select list, the asterisk will be expanded to the following list of columns (in the shown order):

- » All the common columns
- » Every column in the first (left) table that is not a common column
- » Every column in the second (right) table that is not a common column

An asterisk qualified by a table name (for example, COUNTRIES.*) will be expanded to every column of that table that is not a common column.

If a common column is referenced without being qualified by a table name, the column reference points to the column in the first (left) table if the join is an INNER JOIN or a LEFT OUTER JOIN. If it is a RIGHT OUTER JOIN, unqualified references to a common column point to the column in the second (right) table.

NOTE: Splice Machine does not currently support NATURAL SELF JOIN operations.

Examples

If the tables COUNTRIES and CITIES have two common columns named COUNTRY and COUNTRY_ISO_CODE, the following two SELECT statements are equivalent:

```
splice> SELECT *
  FROM COUNTRIES
  NATURAL JOIN CITIES;

splice> SELECT *
  FROM COUNTRIES
  JOIN CITIES
  USING (COUNTRY, COUNTRY_ISO_CODE);
```

The following example is similar to the one above, but it also preserves unmatched rows from the first (left) table:

```
splice> SELECT *
  FROM COUNTRIES
  NATURAL LEFT JOIN CITIES;
```

See Also

- » [JOIN operations](#)
- » [TABLE expression](#)
- » [USING clause](#)

RIGHT OUTER JOIN

A RIGHT OUTER JOIN is one of the [JOIN](#) operations that allow you to specify a JOIN clause. It preserves the unmatched rows from the second (right) table, joining them with a NULL in the shape of the first (left) table. A Right Outer JOIN B is equivalent to B RIGHT OUTER JOIN A, with the columns in a different order.

Syntax

```
TableExpression
{
  ON booleanExpression | USING clause
}
```

The scope of expressions in the ON clause includes the current tables and any tables in query blocks outer to the current SELECT. The ON clause can reference tables not being joined and does not have to reference either of the tables being joined (though typically it does).

Example 1

```
-- get all countries and corresponding cities, including
-- countries without any cities
splice> SELECT COUNTRIES.COUNTRY, CITIES.CITY_NAME
  FROM CITIES  RIGHT OUTER JOIN COUNTRIES
  ON CITIES.COUNTRY_ISO_CODE = COUNTRIES.COUNTRY_ISO_CODE;

-- get all countries in Africa and corresponding cities,
-- including countries without any cities
splice> SELECT COUNTRIES.COUNTRY, CITIES.CITY_NAME
  FROM CITIES
  RIGHT OUTER JOIN COUNTRIES
  ON CITIES.COUNTRY_ISO_CODE = COUNTRIES.COUNTRY_ISO_CODE
  WHERE Countries.region = 'Africa';

-- use the synonymous syntax, RIGHT JOIN, to achieve exactly
-- the same results as in the example above
splice> SELECT COUNTRIES.COUNTRY, CITIES.CITY_NAME
  FROM CITIES
  RIGHT JOIN COUNTRIES
  ON CITIES.COUNTRY_ISO_CODE = COUNTRIES.COUNTRY_ISO_CODE
  WHERE Countries.region = 'Africa';
```

Example 2

```
-- a TableExpression can be a joinOperation. Therefore  
-- you can have multiple join operations in a FROM clause  
-- List every employee number and last name  
-- with the employee number and last name of their manager  
splice> SELECT E.EMPNO, E.LASTNAME, M.EMPNO, M.LASTNAME  
      FROM EMPLOYEE E RIGHT OUTER JOIN  
           DEPARTMENT RIGHT OUTER JOIN EMPLOYEE M  
      ON MGRNO = M.EMPNO  
      ON E.WORKDEPT = DEPTNO;
```

See Also

- » [JOIN operations](#)
- » [TABLE expression](#)
- » [USING clause](#)

Queries

This section contains the reference documentation for the Splice Machine SQL Queries, in the following topics:

Topic	Description
Query	Creates a virtual table based on existing tables or constants built into tables.
Scalar Subquery	A subquery that returns a single row with a single column.
Table Subquery	A subquery that returns multiple rows.



For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](#).

Query

A **Query** creates a virtual table based on existing tables or constants built into tables.

Syntax

```
{
  ( Query
    [ ORDER BY clause ]
    [ result offset clause ]
    [ fetch first clause ]
  ) |
  Query EXCEPT [ ALL | DISTINCT ] Query |
  Query UNION [ ALL | DISTINCT ] Query |
  VALUES Expression
}
```

You can arbitrarily put parentheses around queries, or use the parentheses to control the order of evaluation of the `UNION` operations. These operations are evaluated from left to right when no parentheses are present.

Duplicates in UNION and EXCEPT ALL results

The `ALL` and `DISTINCT` keywords determine whether duplicates are eliminated from the result of the operation. If you specify the `DISTINCT` keyword, then the result will have no duplicate rows. If you specify the `ALL` keyword, then there may be duplicates in the result, depending on whether there were duplicates in the input. `DISTINCT` is the default, so if you don't specify `ALL` or `DISTINCT`, the duplicates will be eliminated. For example, `UNION` builds an intermediate `ResultSet` with all of the rows from both queries and eliminates the duplicate rows before returning the remaining rows. `UNION ALL` returns all rows from both queries as the result.

Depending on which operation is specified, if the number of copies of a row in the left table is L and the number of copies of that row in the right table is R , then the number of duplicates of that particular row that the output table contains (assuming the `ALL` keyword is specified) is:

- » `UNION`: $(L + R)$.
- » `EXCEPT`: the maximum of $(L - R)$ and 0 (zero).

Examples

Here's a simple `SELECT` expression:

```
SELECT *
  FROM ORG;
```

Here's a SELECT with a subquery:

```
SELECT *
  FROM (SELECT CLASS_CODE FROM CL_SCHED) AS CS;
```

Here's a SELECT with a subquery:

```
SELECT *
  FROM (SELECT CLASS_CODE FROM CL_SCHED) AS CS;
```

Here's a UNION that lists all employee numbers from certain departments who are assigned to specified project numbers:

```
SELECT EMPNO, 'emp'
  FROM EMPLOYEE
 WHERE WORKDEPT LIKE 'E%'
UNION
  SELECT EMPNO, 'emp_act'
    FROM EMP_ACT
   WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112');
```

See Also

- » [ORDER BY clause](#)
- » [SELECT expression](#)
- » [SELECT statement](#)
- » [VALUES expression](#)

Scalar Subquery

A *ScalarSubquery* turns a *SelectExpression* result into a scalar value because it returns only a single row and column value.

Syntax

```
( Query
  [ ORDER BY clause ]
  [ result offset clause ]
  [ fetch first clause ]
)
```

Usage

You can place a *ScalarSubquery* anywhere an *Expression* is permitted. The query must evaluate to a single row with a single column.

Scalar subqueries are also called *expression subqueries*.

Examples

The AVG function always returns a single value; thus, this is a scalar subquery:

```
SELECT NAME, COMM
  FROM STAFF
 WHERE EXISTS
   (SELECT AVG(BONUS + 800)
      FROM EMPLOYEE
     WHERE COMM < 5000
       AND EMPLOYEE.LASTNAME = UPPER(STAFF.NAME)
    );
```

See Also

- » [ORDER BY clause](#)
- » [SELECT expression](#)

Table Subquery

A *TableSubquery* is a subquery that returns multiple rows.

Syntax

```
( Query
  [ ORDER BY clause ]
  [ result offset clause ]
  [ fetch first clause ]
)
```

Usage

Unlike a *ScalarSubquery*, a *TableSubquery* is allowed only:

- » as a *TableExpression* in a `FROM` clause
- » with `EXISTS`, `IN`, or quantified comparisons.

When used as a *TableExpression* in a `FROM` clause, or with `EXISTS`, it can return multiple columns.

When used with `IN` or quantified comparisons, it must return a single column.

Example

This example shows a subquery used as a table expression in a `FROM` clause:

```
SELECT VirtualFlightTable.flight_ID
  FROM
    (SELECT flight_ID, orig_airport, dest_airport
      FROM Flights
     WHERE (orig_airport = 'SFO' OR dest_airport = 'SCL')
    )
 AS VirtualFlightTable;
```

This shows one subquery used with `EXISTS` and another used with `IN`:

```
SELECT *
  FROM Flights
 WHERE EXISTS
   (SELECT *
      FROM Flights
     WHERE dest_airport = 'SFO'
     AND orig_airport = 'GRU');

SELECT flight_id, segment_number
  FROM Flights
 WHERE flight_id IN
   (SELECT flight_ID
      FROM Flights
     WHERE orig_airport = 'SFO'
     OR dest_airport = 'SCL');
```

See Also

- » [FROM clause](#)
- » [ORDER BY clause](#)
- » [SELECT expression](#)
- » [TABLE expression](#)

Built-in SQL Functions

This section contains the reference documentation for the SQL Functions that are built into Splice Machine, which are grouped into the following subsections:

- » Conversion Functions
- » Current Session Functions
- » Date and Time Functions
- » Miscellaneous Functions
- » Numeric Functions
- » String Functions
- » Trigonometric Functions
- » Window and Aggregate Functions

Conversion Functions

These are the built-in conversion functions:

Function Name	Description
BIGINT	Returns a 64-bit integer representation of a number or character string in the form of an integer constant.
CAST	Converts a value from one data type to another and provides a data type to a dynamic parameter (?) or a NULL value.
CHAR	Returns a fixed-length character string representation.
DOUBLE	Returns a floating-point number
INTEGER	Returns an integer representation of a number or character string in the form of an integer constant.
SMALLINT	Returns a small integer representation of a number or character string in the form of a small integer constant.
TO_CHAR	Formats a date value into a string.
TO_DATE	Formats a date string according to a formatting specification, and returns a date value.
VARCHAR	Returns a varying-length character string representation of a character string.

Current Session Functions

These are the built-in current session functions:

Function Name	Description
CURRENT_ROLE	Returns the authorization identifier of the current role.
CURRENT_SCHEMA	Returns the schema name used to qualify unqualified database object references.
CURRENT_USER	Depending on context, returns the authorization identifier of either the user who created the SQL session or the owner of the schema.
SESSION_USER	Depending on context, returns the authorization identifier of either the user who created the SQL session or the owner of the schema.
USER	Depending on context, returns the authorization identifier of either the user who created the SQL session or the owner of the schema.

Date and Time Functions

These are the built-in date and time functions:

Function Name	Description
ADD_MONTHS	Returns the date resulting from adding a number of months added to a specified date.
CURRENT_DATE	Returns the current date.
CURRENT_TIME	Returns the current time;
CURRENT_TIMESTAMP	Returns the current timestamp;
DATE	Returns a date from a value.
DAY	Returns the day part of a value.
EXTRACT	Extracts various date and time components from a date expression.
HOUR	Returns the hour part of a value.
LAST_DAY	Returns the date of the last day of the specified month.
MINUTE	Returns the minute part of a value.

MONTH	Returns the numeric month part of a value.
MONTH_BETWEEN	Returns the number of months between two dates.
MONTHNAME	Returns the string month part of a value.
NEXT_DAY	Returns the date of the next specified day of the week after a specified date.
NOW	Returns the current date and time as a timestamp value.
QUARTER	Returns the quarter number (1-4) from a date expression.
SECOND	Returns the seconds part of a value.
TIME	Returns a time from a value.
TIMESTAMP	Returns a timestamp from a value or a pair of values.
TIMESTAMPADD	Adds the value of an interval to a timestamp value and returns the sum as a new timestamp
TIMESTAMPDIFF	Finds the difference between two timestamps, in terms of the specified interval.
TO_CHAR	Formats a date value into a string.
TO_DATE	Formats a date string according to a formatting specification, and returns a date value.
TRUNC or TRUNCATE	Truncates numeric, date, and timestamp values.
WEEK	Returns the year part of a value.
YEAR	Returns the year part of a value.

Miscellaneous Functions

These are the built-in miscellaneous functions:

Function Name	Description
COALESCE	Takes two or more compatible arguments and Returns the first argument that is not null.
NULLIF	Returns NULL if the two arguments are equal, and it Returns the first argument if they are not equal.
NVL	Takes two or more compatible arguments and Returns the first argument that is not null.

ROWID	A <i>pseudocolumn</i> that uniquely defines a single row in a database table.
-------	---

Numeric Functions

These are the built-in numeric functions:

Function Name	Description
ABS or ABSVAL	Returns the absolute value of a numeric expression.
CEIL or CEILING	Round the specified number up, and return the smallest number that is greater than or equal to the specified number.
EXP	Returns e raised to the power of the specified number.
FLOOR	Rounds the specified number down, and Returns the largest number that is less than or equal to the specified number.
LN or LOG	Return the natural logarithm (base e) of the specified number.
LOG10	Returns the base-10 logarithm of the specified number.
MOD	Returns the remainder (modulus) of one number divided by another.
RAND	Returns a random number given a seed number
RANDOM	Returns a random number.
SIGN	Returns the sign of the specified number.
SQRT	Returns the square root of a floating point number;
TRUNC or TRUNCATE	Truncates numeric, date, and timestamp values.

String Functions

These are the built-in string functions:

Function Name	Description
---------------	-------------

Concatenate	Concatenates a character string value onto the end of another character string. Can also be used on bit string values.
INITCAP	Converts the first letter of each word in a string to uppercase, and converts any remaining characters in each word to lowercase.
INSTR	Returns the index of the first occurrence of a substring in a string.
LCASE or LOWER	Takes a character expression as a parameter and Returns a string in which all alpha characters have been converted to lowercase.
LENGTH	Applied to either a character string expression or a bit string expression and Returns the number of characters in the result.
LOCATE	Used to search for a string within another string.
LTRIM	Removes blanks from the beginning of a character string expression.
REGEXP_LIKE	Returns true if a string matches a regular expression.
REPLACE	Replaces all occurrences of a substring with another substring
RTRIM	Removes blanks from the end of a character string expression.
SUBSTR	Return a portion of string beginning at the specified position for the number of characters specified or rest of the string.
TRIM	Takes a character expression and Returns that expression with leading and/or trailing pad characters removed.
UCASE or UPPER	Takes a character expression as a parameter and Returns a string in which all alpha characters have been converted to uppercase.

Trigonometric Functions

These are the built-in trigonometric functions:

Function Name	Description
ACOS	Returns the arc cosine of a specified number.
ASIN	Returns the arc sine of a specified number.
ATAN	Returns the arc tangent of a specified number.
ATAN2	Returns the arctangent, in radians, of the quotient of the two arguments.

COS	Returns the cosine of a specified number.
COSH	Returns the hyperbolic cosine of a specified number.
COT	Returns the cotangens of a specified number.
DEGREES	Converts a specified number from radians to degrees.
PI	Returns a value that is closer than any other value to pi.
RADIANS	Converts a specified number from degrees to radians.
SIN	Returns the sine of a specified number.
SINH	Returns the hyperbolic sine of a specified number.
TAN	Returns the tangent of a specified number.
TANH	Returns the hyperbolic tangent of a specified number

Window and Aggregate Functions

These are the built-in window and aggregate functions:

Function Name	Description
AVG	Returns the average computed over a subset (partition) of a table.
COUNT	Returns the number of rows in a partition.
DENSE_RANK	Returns the ranking of a row within a partition.
FIRST_VALUE	Returns the first value within a partition..
LAG	Returns the value of an expression evaluated at a specified offset number of rows <i>before</i> the current row in a partition.
LAST_VALUE	Returns the last value within a partition..
LEAD	Returns the value of an expression evaluated at a specified offset number of rows <i>after</i> the current row in a partition.
MAX	Returns the maximum value computed over a partition.
MIN	Returns the minimum value computed over a partition.

Function Name	Description
RANK	Returns the ranking of a row within a subset of a table.
ROW_NUMBER	Returns the row number of a row within a partition.
STDDEV_POP	Returns the population standard deviation of a set of numeric values
STDDEV_SAMP	Returns the sample standard deviation of a set of numeric values
SUM	Returns the sum of a value calculated over a partition.



For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](#).

ABS or ABSVAL

ABS or ABSVAL returns the absolute value of a numeric expression.

Syntax

```
ABS(NumericExpression)
```

NumericExpression

A numeric expression; all built-in numeric types are supported: [SMALLINT](#)

Results

The return type is the type of the input parameter.

Example

```
splice> VALUES ABS(-3);
1
-----
3
1 row selected
```

See Also

» [About Data Types](#)

ACOS

The ACOS function returns the arc cosine of a specified number.

Syntax

```
ACOS ( number )
```

number

A [DOUBLE PRECISION](#) number that specifies the cosine, in radians, of the angle that you want.

Results

The data type of the returned value is a [DOUBLE PRECISION](#) number. The returned value, in radians, is in the range of zero (0) to pi.

- » If the specified *number* is NULL, the result of this function is NULL.
- » If the absolute value of the specified number is greater than 1, an exception is returned that indicates that the value is out of range (SQL state 22003).

Example

```
splice> VALUES ACOS(0.5);
1
-----
1.0471975511965979

1 row selected
```

See Also

- » [DOUBLE PRECISION](#) data type
- » [ASIN](#) function
- » [ATAN](#) function
- » [ATAN2](#) function
- » [COS](#) function
- » [COSH](#) function

- » [COT function](#)
- » [DEGREES function](#)
- » [RADIANS function](#)
- » [SIN function](#)
- » [SINH function](#)
- » [TAN function](#)
- » [TANH function](#)

ADD_MONTHS

The ADD_MONTHS function returns the date resulting from adding a number of months added to a specified date.

Syntax

```
ADD_MONTHS(Date source, int numOfMonths);
```

source

The source date. This can be a DATE value, or any value that can be implicitly converted to DATE.

numOfMonths

An integer value that specifies the number of months to add to the source date.

Results

The returned string always has data type DATE.

If date is the last day of the month or if the resulting month has fewer days than the day component of date, then the result is the last day of the resulting month. Otherwise, the result has the same day component as date.

Examples

```
splice> VALUES(ADD_MONTHS(CURRENT_DATE,5));
1
-----
2015-02-22
1 row selected

splice> VALUES(ADD_MONTHS(CURRENT_DATE,-5));
1
-----
2014-04-22
1 row selected

splice> VALUES(ADD_MONTHS(DATE(CURRENT_TIMESTAMP),-5));
1
-----
2014-04-22
1 row selected

splice> VALUES(ADD_MONTHS(DATE('2014-01-31'),1));
1
-----
2014-02-28
1 row selected
```

ASIN

The ASIN function returns the arc sine of a specified number.

Syntax

```
ASIN ( number )
```

number

A [DOUBLE PRECISION](#) number that specifies the sine, in radians, of the angle that you want.

Results

The data type of the returned value is a [DOUBLE PRECISION](#) number. The returned value, in radians, is in the range pi/2 to -pi/2.

- » If the specified number is NULL, the result of this function is NULL.
- » If the specified number is zero (0), the result of this function is zero with the same sign as the specified number.
- » If the absolute value of the specified number is greater than 1, an exception is returned that indicates that the value is out of range (SQL state 22003).

Example

```
splice> VALUES ASIN(0.5);
1
-----
0.5235987755982989
1 row selected
```

See Also

- » [DOUBLE PRECISION](#) data type
- » [ACOS](#) function
- » [ATAN](#) function
- » [ATAN2](#) function
- » [COS](#) function

- » [COSH function](#)
- » [COT function](#)
- » [DEGREES function](#)
- » [RADIANS function](#)
- » [SIN function](#)
- » [SINH function](#)
- » [TAN function](#)
- » [TANH function](#)

ATAN

The ATAN function returns the arc tangent of a specified number.

Syntax

```
ATAN ( number )
```

number

A [DOUBLE PRECISION](#) number that specifies the tangent, in radians, of the angle that you want.

Results

The data type of the returned value is a [DOUBLE PRECISION](#) number. The returned value, in radians, is in the range $\pi/2$ to $-\pi/2$.

- » If the specified number is `NULL`, the result of this function is `NULL`.
- » If the specified number is zero (0), the result of this function is zero with the same sign as the specified number.
- » If the absolute value of the specified number is greater than 1, an exception is returned that indicates that the value is out of range (SQL state 22003).

Example

```
splice> VALUES ATAN(0.5);
1
-----
0.46364760900008061
1 row selected
```

See Also

- » [DOUBLE PRECISION](#) data type
- » [ACOS](#) function
- » [ASIN](#) function
- » [ATAN2](#) function
- » [COS](#) function

- » [COSH function](#)
- » [COT function](#)
- » [DEGREES function](#)
- » [RADIANS function](#)
- » [SIN function](#)
- » [SINH function](#)
- » [TAN function](#)
- » [TANH function](#)

ATAN2

The ATAN2 function returns the arctangent, in radians, of the quotient of the two arguments.

Syntax

```
ATAN2 ( y, x )
```

y

A **DOUBLE PRECISION** number.

x

A **DOUBLE PRECISION** number.

Results

ATAN2 returns the arc tangent of y/x in the range $-pi$ to pi radians, as a **DOUBLE PRECISION** number.

- » If either argument is **NULL**, the result of the function is **NULL**.
- » If the first argument is zero and the second argument is positive, the result of the function is zero.
- » If the first argument is zero and the second argument is negative, the result of the function is the double value closest to pi .
- » If the first argument is positive and the second argument is zero, the result is the double value closest to $pi/2$.
- » If the first argument is negative and the second argument is zero, the result is the double value closest to $-pi/2$.

Example

```
splice> VALUES ATAN2(1, 0);
1
-----
1.5707963267948966
1 row selected
```

See Also

- » [DOUBLE PRECISION](#) data type
- » [ACOS](#) function

- » [ASIN function](#)
- » [ATAN function](#)
- » [COS function](#)
- » [COSH function](#)
- » [COT function](#)
- » [DEGREES function](#)
- » [RADIANS function](#)
- » [SIN function](#)
- » [SINH function](#)
- » [TAN function](#)
- » [TANH function](#)

AVG

AVG evaluates the average of an expression over a set of rows. You can use it as an window (analytic) function.

Syntax

```
AVG ( [ DISTINCT | ALL ] Expression )
```

DISTINCT

If this qualifier is specified, duplicates are eliminated

ALL

If this qualifier is specified, all duplicates are retained. This is the default value.

Expression

An expression that evaluates to a numeric data type: [SMALLINT](#).

The expression can contain multiple column references or expressions, but it cannot contain another aggregate or subquery, and it must evaluate to an ANSI SQL numeric data type. This means that you can call methods that evaluate to ANSI SQL data types.

If an expression evaluates to NULL, the aggregate skips that value.

Usage

Only one DISTINCT aggregate expression per *Expression* is allowed. For example, the following query is not valid:

```
--- query not valid
SELECT AVG (DISTINCT AtBats), SUM (DISTINCT Hits)
  FROM Batting;
```

NOTE: Note that specifying DISTINCT can result in a different value, since a smaller number of values may be averaged. For example, if a column contains the values 1.0, 1.0, 1.0, 1.0, and 2.0, AVG(col) returns a smaller value than AVG(DISTINCT col).

Results

The resulting data type is the same as the expression on which it operates; it will never overflow.

The following query, for example, returns the INTEGER 1, which might not be what you would expect:

```
SELECT AVG(c1)
  FROM (VALUES (1), (1), (1), (1), (2))
AS myTable (c1);
```

[CAST](#) the expression to another data type if you want more precision:

```
SELECT AVG(CAST (c1 AS DOUBLE PRECISION))
  FROM (VALUES (1), (1), (1), (1), (2)) AS myTable (c1);
```

Aggregate Example

```
splice> SELECT AVG(salary) "Average" FROM Salaries;
Average
-----
2949737

1 row selected
```

Analytic Example

The following example shows the average salary paid, per position, for the San Francisco Giants in 2015:

```
splice> SELECT Position, Players.ID, Salary, AVG(Cast(Salary as DECIMAL(11,3)))
      OVER (PARTITION by Position) "Average for Position"
      FROM players join Salaries on players.ID=salaries.ID
      WHERE Team='Giants' and Season=2015;
```

POS&	ID	SALARY	Average for Po&
C	1	17277777	3733139.8000
C	13	468674	3733139.8000
C	18	800000	3733139.8000
C	20	41598	3733139.8000
C	24	77650	3733139.8000
IF	23	91516	91516.0000
1B	2	3600000	1815252.5000
1B	26	30505	1815252.5000
LF	6	4000000	1792987.0000
LF	16	278961	1792987.0000
LF	27	1100000	1792987.0000
P	28	6950000	4759511.3333
P	29	485314	4759511.3333
P	30	4000000	4759511.3333
P	31	12000000	4759511.3333
P	32	9000000	4759511.3333
P	33	18000000	4759511.3333
P	34	20833333	4759511.3333
P	35	3578825	4759511.3333
P	36	2100000	4759511.3333
P	37	210765	4759511.3333
P	38	507500	4759511.3333
P	39	507500	4759511.3333
P	40	6000000	4759511.3333
P	41	6000000	4759511.3333
P	42	374385	4759511.3333
P	43	4000000	4759511.3333
P	44	72103	4759511.3333
P	45	91516	4759511.3333
P	46	5000000	4759511.3333
P	47	74877	4759511.3333
P	48	163620	4759511.3333
3B	5	509000	2654500.0000
3B	14	4800000	2654500.0000
MI	15	288767	288767.0000
SS	4	3175000	3175000.0000
RF	8	18500000	9166666.6666
RF	10	1000000	9166666.6666
RF	12	8000000	9166666.6666
2B	3	507500	339719.5000
2B	11	171939	339719.5000
CF	7	10250000	10250000.0000
UT	17	1450000	749959.0000
UT	22	49918	749959.0000

```
OF | 9      | 3600000          | 962397.2500
OF | 19     | 91516            | 962397.2500
OF | 21     | 149754           | 962397.2500
OF | 25     | 8319             | 962397.2500
```

```
48 rows selected
```

See Also

- » [About Data Types](#)
- » [Window and aggregate functions](#)
- » [COUNT function](#)
- » [MAX function](#)
- » [MIN function](#)
- » [SUM function](#)
- » [OVER clause](#)
- » [Using Window Functions](#) in the *Developer Guide*.

BIGINT

The BIGINT function returns a 64-bit integer representation of a number or character string in the form of an integer constant.

Syntax

```
BIGINT (CharacterExpression | NumericExpression )
```

CharacterExpression

An expression that returns a character string value of length not greater than the maximum length of a character constant. Leading and trailing blanks are eliminated and the resulting string must conform to the rules for forming an SQL integer constant. The character string cannot be a long string. If the argument is a CharacterExpression, the result is the same number that would occur if the corresponding integer constant were assigned to a big integer column or variable.

NumericExpression

An expression that returns a value of any built-in numeric data type. If the argument is a NumericExpression, the result is the same number that would occur if the argument were assigned to a big integer column or variable. If the whole part of the argument is not within the range of integers, an error occurs. The decimal part of the argument is truncated if present.

Results

The result of the function is a big integer.

If the argument can be NULL, the result can be NULL; if the argument is NULL, the result is the NULL value.

Example

Using the Batting table from our Doc Examples database, select the TotalBases column in big integer form for further processing in the application:

```
splice> SELECT ID, BIGINT(TotalBases) "TotalBases"
      FROM Batting
     WHERE ID < 11;
ID      |TotalBases
-----
1       |262
2       |235
3       |174
4       |234
5       |245
6       |135
7       |170
8       |99
9       |135
10      |85
10 rows selected
```

See Also

- » [About Data Types](#)
- » [BIGINT data type](#)

CAST

The CAST function converts a value from one data type to another and provides a data type to a dynamic parameter or a NULL value.

CAST expressions are permitted anywhere expressions are permitted.

Syntax

```
CAST ( [ Expression | NULL | ? ]
      AS Datatype)
```

The data type to which you are casting an expression is the *target type*. The data type of the expression from which you are casting is the *source type*.

CAST conversions among ANSI SQL data types

The following table shows valid explicit conversions between source types and target types for SQL data types. This table shows which explicit conversions between data types are valid. The first column on the table lists the source data types. The first row lists the target data types. A "Y" indicates that a conversion from the source to the target is valid. For example, the first cell in the second row lists the source data type SMALLINT. The remaining cells on the second row indicate the whether or not you can convert SMALLINT to the target data types that are listed in the first row of the table.

TYPES	B	S	I	D	D	F	V	L						T		
	O	M	N	B	E	O	A	O	V	C	C	C	C	IMES		
O	L	T	I	G	I	R	B	L	A	H	H	A	B	STAMP		
E	I	G	I	M	E	R	B	O	A	H	H	A	B	TIME		
A	N	E	N	A	N	M	A	L	T	A	H	A	B	DATE		
N	T	R	T	A	L	A	L	E	E	A	R	A	B	TIME		
BOOLEAN	Y	-	-	-	-	-	-	-	Y	Y	Y	Y	-	-	-	-
SMALLINT	-	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-
INTEGER	-	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-	-
BIGINT	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	-	-

TYPES	B	M	I	B	D	D	F	C	V	L			T
	O	O	N	E	E	O	L	A	A	O			M
E	E	G	I	M	R	D	F	C	V				S
A	A	I	N	M	E	O	L	A	A				E
N	N	R	T	L	A	R	T	R	R				T
DECIMAL	-	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-
REAL	-	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-
DOUBLE	-	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-
FLOAT	-	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-
CHAR	Y	Y	Y	Y	Y	-	-	-	Y	Y	Y	Y	Y
VARCHAR	Y	Y	Y	Y	Y	-	-	-	Y	Y	Y	Y	Y
LONG VARCHAR	Y	-	-	-	-	-	-	-	Y	Y	Y	Y	-
CLOB	Y	-	-	-	-	-	-	-	Y	Y	Y	Y	-
BLOB	-	-	-	-	-	-	-	-	-	-	-	Y	-
DATE	-	-	-	-	-	-	-	-	Y	Y	-	-	-
TIME	-	-	-	-	-	-	-	-	Y	Y	-	-	Y
TIMESTAMP	-	-	-	-	-	-	-	-	Y	Y	-	-	Y

If a conversion is valid, CASTs are allowed. Size incompatibilities between the source and target types might cause runtime errors.

Type Categories

This section lists information about converting specific data types. The Splice Machine ANSI SQL data types are categorized as follows:

Category	Data Types
<i>logical</i>	BOOLEAN
<i>numeric</i>	Exact numeric: SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC Approximate numeric: FLOAT, REAL, DOUBLE PRECISION
<i>string</i>	Character string: CLOB, CHAR, VARCHAR, LONG VARCHAR Bit string: BLOB
<i>date and time</i>	DATE, TIME, TIMESTAMP

Conversion Notes

This section lists additional information about casting of certain data types.

Applying Multiple Conversions

As shown in the above table, you cannot convert freely among all types. For example, you cannot CAST an INTEGER value to a VARCHAR value. However, you may be able to achieve your conversion by using multiple CAST operations.

For example, since you can convert an INTEGER value to a CHAR value, and you can convert a CHAR value to a VARCHAR value, you can use multiple CAST operations, as shown here:

```
CAST(CAST(123 AS CHAR(10)) AS VARCHAR(10));SELECT CAST(CAST(myId as CHAR(20)) as VARCHAR(20));
```

Conversions to and from logical types

These notes apply to converting logical values to strings and vice-versa:

- » A BOOLEAN value can be cast explicitly to any of the string types. The result is 'true', 'false', or null.
- » Conversely, string types can be cast to BOOLEAN; however, an error is raised if the string value is not 'true', 'false', 'unknown', or null.
- » Casting 'false' to BOOLEAN results in a null value.

Conversions from numeric types

A numeric type can be converted to any other numeric type. These notes apply:

- » If the target type cannot represent the non-fractional component without truncation, an exception is raised.

- » If the target numeric cannot represent the fractional component (scale) of the source numeric, then the source is silently truncated to fit into the target. For example, casting 763.1234 as INTEGER yields 763.

Conversions from and to bit strings

Bit strings can be converted to other bit strings, but not to character strings. Strings that are converted to bit strings are padded with trailing zeros to fit the size of the target bit string. The BLOB type is more limited and requires explicit casting. In most cases the BLOB type cannot be cast to and from other types: you can cast a BLOB only to another BLOB, but you can cast other bit string types to a BLOB.

Conversions of date/time values

A date/time value can always be converted to and from a TIMESTAMP.

If a DATE is converted to a TIMESTAMP, the TIME component of the resulting TIMESTAMP is always 00:00:00.

If a TIME data value is converted to a TIMESTAMP, the DATE component is set to the value of CURRENT_DATE at the time the CAST is executed.

If a TIMESTAMP is converted to a DATE, the TIME component is silently truncated.

If a TIMESTAMP is converted to a TIME, the DATE component is silently truncated.

Examples

```
splice> SELECT CAST (TotalBases AS BIGINT)
      FROM Batting;

      -- convert timestamps to text
splice> INSERT INTO mytable (text_column)
      VALUES (CAST (CURRENT_TIMESTAMP AS VARCHAR(100)));

      -- you must cast NULL as a data type to use it
splice> SELECT airline
      FROM Airlines
      UNION ALL
      VALUES (CAST (NULL AS CHAR(2)));

      -- cast a double as a decimal
splice> SELECT CAST (FLYING_TIME AS DECIMAL(5,2))
      FROM FLIGHTS;

      -- cast a SMALLINT to a BIGINT
splice> VALUES CAST (CAST (12 as SMALLINT) as BIGINT);
```

See Also

- » About Data Types

CEIL or CEILING

The CEIL and CEILING functions round the specified number up, and return the smallest number that is greater than or equal to the specified number.

Syntax

```
CEIL ( number )
```

```
CEILING ( number )
```

number

A [DOUBLE PRECISION](#) value.

The expression can contain multiple column references or expressions, but it cannot contain another aggregate or subquery, and it must evaluate to an ANSI SQL numeric data type. This means that you can call methods that evaluate to ANSI SQL data types.

If an expression evaluates to `NULL`, the aggregate skips that value.

Results

The data type of the returned value is a [DOUBLE PRECISION](#) number.

The returned value is the smallest (closest to negative infinity) double floating point value that is greater than or equal to the specified number. The returned value is equal to a mathematical integer.

- » If the specified number is `NULL`, the result of these functions is `NULL`.
- » If the specified number is equal to a mathematical integer, the result of these functions is the same as the specified number.
- » If the specified number is zero (0), the result of these functions is zero.
- » If the specified number is less than zero but greater than -1.0, then the result of these functions is zero.

Example

```
splice> VALUES CEIL(3.33);
1
-----
4

1 row selected

splice> VALUES CEILING(3.67);
1
-----
4

1 row selected
```

See Also

» [DOUBLE PRECISION](#) data type

CHAR

The CHAR function returns a fixed-length character string representation. The representations are:

- » A character string, if the first argument is any type of character string.
- » A datetime value, if the first argument is a date, time, or timestamp.
- » A decimal number, if the first argument is a decimal number.
- » A double-precision floating-point number, if the first argument is a DOUBLE or REAL.
- » An integer number, if the first argument is a SMALLINT, INTEGER, or BIGINT.

The first argument must be of a built-in data type.

The result of the CHAR function is a fixed-length character string. If the first argument can be NULL, the result can be NULL. If the first argument is NULL, the result is the NULL value.

Character to character syntax

```
CHAR (CharacterExpression [, integer] )
```

CharacterExpression

An expression that returns a value that is CHAR, VARCHAR, LONG VARCHAR, or CLOB data type.

integer

The length attribute for the resulting fixed length character string. The value must be between 0 and 254.

Results

If the length of the character-expression is less than the length attribute of the result, the result is padded with blanks up to the length of the result.

If the length of the character-expression is greater than the length attribute of the result, truncation is performed. A warning is returned unless the truncated characters were all blanks and the character-expression was not a long string (LONG VARCHAR or CLOB).

Integer to character syntax

```
CHAR (IntegerExpression )
```

IntegerExpression

An expression that returns a value that is an integer data type (either SMALLINT, INTEGER or BIGINT).

Results

The result is the character string representation of the argument in the form of an SQL integer constant. The result consists of n characters that are the significant digits that represent the value of the argument with a preceding minus sign if the argument is negative. It is left justified.

- » If the first argument is a small integer: the length of the result is 6. If the number of characters in the result is less than 6, then the result is padded on the right with blanks to length 6.
- » If the first argument is a large integer: the length of the result is 11. If the number of characters in the result is less than 11, then the result is padded on the right with blanks to length 11.
- » If the first argument is a big integer: the length of the result is 20. If the number of characters in the result is less than 20, then the result is padded on the right with blanks to length 20.

Datetime to character syntax

```
CHAR (DatetimeExpression )
```

DatetimeExpression

An expression that is one of the following three data types:

Type	Description
DATE	The result is the character representation of the date. The length of the result is 10.
TIME	The result is the character representation of the time. The length of the result is 8.
TIMESTAMP	The result is the character string representation of the timestamp. The length of the result is 26.

Decimal to character

```
CHAR (DecimalExpression )
```

DecimalExpression

An expression that returns a value that is a decimal data type.

If a different precision and scale is desired, you can use the DECIMAL scalar function first to make the change.

Floating point to character syntax

```
CHAR (FloatingPointExpression )
```

FloatingPointExpression

An expression that returns a value that is a floating-point data type (DOUBLE or REAL).

Example

Use the CHAR function to return the values for PlateAppearances (defined as smallint) as a fixed length character string:

```
splice> SELECT CHAR(AtBats * 2) "DoubledAtBats"
   FROM Batting WHERE ID <= 10;
DoubledAtBats
-----
1246
1112
864
1122
1224
784
1102
446
744
548

10 rows selected
```

Since AtBats is declared as SMALLINT in our Examples database, each of the resulting values is padded with blank characters to make it 6 characters long.

See Also

- » [About Data Types](#)

COALESCE

The COALESCE function returns the first non-NULL expression from a list of expressions.

You can also use COALESCE as a variety of a CASE expression. For example:

```
COALESCE( expression_1, expression_2,...expression_n);
```

is equivalent to:

```
CASE WHEN expression_1 IS NOT NULL THEN expression_1
      ELSE WHEN expression_1 IS NOT NULL THEN expression_2
      ...
      ELSE expression_n;
```

Syntax

```
COALESCE ( expression1, expression2 [, expressionN]* )
```

expression1

An expression.

expression1

An expression.

expressionN

You can specify more than two arguments; **you MUST specify at least two arguments.**

Usage

VALUE is a synonym for COALESCE that is accepted by Splice Machine, but is not recognized by the SQL standard.

Results

The result is NULL only if all of the arguments are NULL.

An error occurs if all of the parameters of the function call are dynamic.

Example

```
-- create table with three different integer types
splice> SELECT ID, FldGames, PassedBalls, WildPitches, Pickoffs,
COALESCE(PassedBalls, WildPitches, Pickoffs) as "FirstNonNull"
FROM Fielding
WHERE FldGames>50
ORDER BY ID;
ID      |FLDGA&|PASSE&|WILDP&|PICKO&|First&
-----
1       | 142   | 4      | 20     | 0      | 4
2       | 131   | NULL   | NULL   | NULL   | NULL
3       | 99    | NULL   | NULL   | NULL   | NULL
4       | 140   | NULL   | NULL   | NULL   | NULL
5       | 142   | NULL   | NULL   | NULL   | NULL
6       | 88    | NULL   | NULL   | NULL   | NULL
7       | 124   | NULL   | NULL   | NULL   | NULL
8       | 51    | NULL   | NULL   | NULL   | NULL
9       | 93    | NULL   | NULL   | NULL   | NULL
10      | 79    | NULL   | NULL   | NULL   | NULL
39      | 73    | NULL   | NULL   | 0      | 0
40      | 52    | NULL   | NULL   | 0      | 0
41      | 70    | NULL   | NULL   | 2      | 2
42      | 55    | NULL   | NULL   | 0      | 0
43      | 77    | NULL   | NULL   | 0      | 0
46      | 67    | NULL   | NULL   | 0      | 0
49      | 134   | 4      | 34     | 2      | 4
50      | 119   | NULL   | NULL   | NULL   | NULL
51      | 147   | NULL   | NULL   | NULL   | NULL
52      | 148   | NULL   | NULL   | NULL   | NULL
53      | 152   | NULL   | NULL   | NULL   | NULL
54      | 64    | NULL   | NULL   | NULL   | NULL
55      | 93    | NULL   | NULL   | NULL   | NULL
56      | 147   | NULL   | NULL   | NULL   | NULL
57      | 85    | NULL   | NULL   | NULL   | NULL
58      | 62    | NULL   | NULL   | NULL   | NULL
59      | 64    | NULL   | NULL   | NULL   | NULL
62      | 53    | 1      | 11     | 0      | 1
64      | 59    | NULL   | NULL   | NULL   | NULL
81      | 76    | NULL   | NULL   | 0      | 0
82      | 71    | NULL   | NULL   | 1      | 1
84      | 68    | NULL   | NULL   | 0      | 0
92      | 81    | NULL   | NULL   | 3      | 3
33 rows selected
```

Concatenation Operator

The concatenation operator, `||`, concatenates its right operand onto the end of its left operand; it operates on character string or bit string expressions.

NOTE: Since all built-in data types are implicitly converted to strings, this function can act on all built-in data types.

Syntax

```
{
  { CharacterExpression || CharacterExpression } |
  { BitExpression || BitExpression }
}
```

CharacterExpression

An expression.

expression1

An expression.

expressionN

You can specify more than two argument; you MUST specify at least two arguments.

Results

For character strings:

- » If both the left and right operands are of type `VARCHAR`.
- » The normal blank padding/trimming rules for `CHAR` and `VARCHAR` apply to the result of this operator.
- » The length of the resulting string is the sum of the lengths of both operands.

Examples

```
-- returns 'San Francisco Giants'
splice> VALUES 'San' || ' ' || 'Francisco' || ' ' || 'Giants';

-- returns NULL
splice> VALUES CAST (null AS VARCHAR(7))|| 'Something';

-- returns 'Today it is: 93'
splice> VALUES 'Today it is: ' || '93';
```

See Also

- » [About Data Types](#)
- » [INITCAP function](#)
- » [INSTR function](#)
- » [LCASE function](#)
- » [LENGTH function](#)
- » [LTRIM function](#)
- » [REGEX_LIKE operator](#)
- » [REPLACE function](#)
- » [RTRIM function](#)
- » [SUBSTR function](#)
- » [TRIM function](#)
- » [UCASE function](#)

COS

The COS function returns the cosine of a specified number.

Syntax

```
COS ( number )
```

number

A **DOUBLE PRECISION** number that specifies the angle, in radians, for which you want the cosine computed.

Results

The data type of the returned value is a **DOUBLE PRECISION** number.

If input argument is **NULL**, the result of the function is **NULL**.

Example

```
splice> VALUES COS(84.4);
1
-----
-0.9118608758306834

1 row selected
```

See Also

- » [DOUBLE PRECISION](#) data type
- » [ACOS](#) function
- » [ASIN](#) function
- » [ATAN](#) function
- » [ATAN2](#) function
- » [COSH](#) function
- » [COT](#) function
- » [DEGREES](#) function

» [RADIAN function](#)

» [SIN function](#)

» [SINH function](#)

» [TAN function](#)

» [TANH function](#)

COSH

The `COSH` function returns the hyperbolic cosine of a specified number.

Syntax

```
COSH ( number )
```

number

A `DOUBLE PRECISION` number that specifies the angle, in radians, for which you want the hyperbolic cosine computed.

Results

The data type of the returned value is a `DOUBLE PRECISION` number.

- » If the specified number is `NULL`, the result of this function is `NULL`.
- » If the specified number is zero (0), the result of this function is one (1.0).

Example

```
splice> VALUES COSH(1.234);
1
-----
2.2564425307671042E36
1 row selected
```

See Also

- » [DOUBLE PRECISION](#) data type
- » [ACOS](#) function
- » [ASIN](#) function
- » [ATAN](#) function
- » [ATAN2](#) function
- » [COS](#) function
- » [COT](#) function

- » [DEGREES function](#)
- » [RADIAN function](#)
- » [SIN function](#)
- » [SINH function](#)
- » [TAN function](#)
- » [TANH function](#)

COT

The COT function returns the cotangent of a specified number.

Syntax

```
COT ( number )
```

number

A [DOUBLE PRECISION](#) number that specifies the angle, in radians, for which you want the cotangent computed.

Results

The data type of the returned value is a [DOUBLE PRECISION](#) number.

- » If the specified number is NULL, the result of this function is NULL.
- » If the specified number is zero (0), the result of this function is one (1.0).

Example

```
splice> VALUES COT(1.234);
1
-----
0.35013639786791445

1 row selected
```

See Also

- » [DOUBLE PRECISION](#) data type
- » [ACOS](#) function
- » [ASIN](#) function
- » [ATAN](#) function
- » [ATAN2](#) function
- » [COS](#) function
- » [COSH](#) function

- » [DEGREES function](#)
- » [RADIAN function](#)
- » [SIN function](#)
- » [SINH function](#)
- » [TAN function](#)
- » [TANH function](#)

COUNT

COUNT returns the number of rows returned by the query. You can use it as an window (analytic) function.

The COUNT(*Expression*) version returns the number of row where *Expression* is not null. You can count either all rows, or only distinct values of *Expression*.

The COUNT(*) version returns all rows, including duplicates and nulls.

Syntax

```
COUNT( [ DISTINCT | ALL ] Expression )
```

DISTINCT

If this qualifier is specified, duplicates are eliminated from the count.

ALL

If this qualifier is specified, all duplicates are retained. This is the default value.

Expression

An expression that evaluates to a numeric data type: [SMALLINT](#).

An *Expression* can contain multiple column references or expressions, but it cannot contain another aggregate or subquery.

If an *Expression* evaluates to NULL, the aggregate skips that value.

Usage

Only one DISTINCT aggregate expression per *Expression* is allowed. For example, the following query is not valid:

```
-- query not allowed
SELECT COUNT (DISTINCT flying_time), SUM (DISTINCT miles)
FROM Flights
```

NOTE: Note that specifying DISTINCT can result in a different value, since a smaller number of values may be counted. For example, if a column contains the values 1, 1, 1, 1, and 2, COUNT(col) returns a greater value than COUNT(DISTINCT col).

Results

The resulting data type is BIGINT.

Aggregate Example

```
splice> Select COUNT (Name) "Players", Team
      FROM Players
      GROUP BY Team
      HAVING COUNT(Team) > 1;
Players          | TEAM
-----
-
46              | Cards
48              | Giants

2 rows selected
```

Analytic Example

The following example shows the product ID, quantity, and count of all rows from the beginning of the data window:

```
splice> SELECT displayName, homeruns,
      COUNT(*) OVER (ORDER BY HomeRuns ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT RO
W)
      as "Running Count"
      FROM Players JOIN Batting ON Players.ID=Batting.ID
      WHERE homeRuns > 5
      ORDER BY "Running Count";
DISPLAYNAME          | HOMERUNS | Running Count
-----
Jeremy Packman      | 6          | 1
Jason Minman        | 7          | 2
Stan Post            | 7          | 3
John Purser          | 8          | 4
Harry Pennello       | 9          | 5
Kelly Wacherman     | 11         | 6
Mitch Duffer         | 12         | 7
Michael Rastono     | 13         | 8
Jack Hellman         | 13         | 9
Jonathan Pearlman   | 17         | 10
Roger Green          | 17         | 11
Billy Bopper         | 18         | 12
Buddy Painter        | 19         | 13
Bob Cranker          | 21         | 14
Mitch Canepa         | 28         | 15

15 rows selected
```

See Also

- » [About Data Types](#)
- » [Window and Aggregate Functions](#)
- » [**AVG** function](#)
- » [**MAX** function](#)
- » [**MIN** function](#)
- » [**SUM** function](#)
- » [**OVER** clause](#)

CURRENT SCHEMA

CURRENT SCHEMA returns the schema name used to qualify unqualified database object references.

NOTE: CURRENT SCHEMA and CURRENT SQLID are synonyms.

Syntax

```
CURRENT SCHEMA
```

– or, alternatively:

```
CURRENT SQLID
```

Results

The returned value is a string with a length of up to 128 characters.

Examples

```
splice> VALUES(CURRENT SCHEMA);
1
-----
SPICE
1 row selected
```

CURRENT_DATE

`CURRENT_DATE` returns the current date.

NOTE: This function returns the same value if it is executed more than once in a single statement, which means that the value is fixed, even if there is a long delay between fetching rows in a cursor.

Syntax

```
CURRENT_DATE
```

or, alternately

```
CURRENT DATE
```

Results

A `DATE` value.

Examples

The following query finds all players older than 33 years (as of Nov. 9, 2015) on the Cards baseball team:

```
splice> SELECT displayName, birthDate
      FROM Players
     WHERE (BirthDate+(33 * 365.25)) <= CURRENT_DATE AND Team='Cards';
DISPLAYNAME          | BIRTHDATE
-----
Yuri Milleton        | 1982-07-13
Jonathan Pearlman    | 1982-05-28
David Janssen         | 1979-08-10
Jason Larrimore      | 1978-10-23
Tam Croonster        | 1980-12-19
Alex Wister           | 1981-08-30
Robert Cohen          | 1975-09-05
Mitch Brandon         | 1980-06-06

8 rows selected
```

See Also

- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)
- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [Working with Dates](#) in the *Developer's Guide*

CURRENT_ROLE

CURRENT_ROLE returns the authorization identifier of the current role. If there is no current role, it returns NULL.

This function returns a string of up to 258 characters. This is twice the length of an identifier ($128*2$) + 2, to allow for quoting.

Syntax

```
CURRENT_ROLE
```

Example

```
splice> VALUES CURRENT_ROLE;
```

See Also

- » [CREATE_ROLE statement](#)
- » [DROP_ROLE statement](#)
- » [GRANT statement](#)
- » [REVOKE statement](#)
- » [SET ROLE statement](#)
- » [SYSROLES system table](#)

CURRENT_TIME

CURRENT_TIME returns the current time.

NOTE: This function returns the same value if it is executed more than once in a single statement, which means that the value is fixed, even if there is a long delay between fetching rows in a cursor.

Syntax

```
CURRENT_TIME
```

or, alternately

```
CURRENT TIME
```

Results

A time value.

Examples

```
splice> VALUES CURRENT_TIME;
1
-----
11:02:57
1 row selected
```

CURRENT_TIMESTAMP

CURRENT_TIMESTAMP returns the current timestamp.

NOTE: This function returns the same value if it is executed more than once in a single statement, which means that the value is fixed, even if there is a long delay between fetching rows in a cursor.

Syntax

```
CURRENT_TIMESTAMP
```

or, alternately

```
CURRENT_TIMESTAMP
```

Results

A timestamp value.

Examples

```
splice> VALUES CURRENT_TIMESTAMP;
1
-----
2015-11-19 11:03:44.095
1 row selected
```

CURRENT_USER

When used outside stored routines, CURRENT_USER, [USER](#), and [SESSION_USER](#) all return the authorization identifier of the user who created the SQL session.

SESSION_USER also always returns this value when used within stored routines.

If used within a stored routine created with EXTERNAL SECURITY DEFINER, however, CURRENT_USER and [USER](#) return the authorization identifier of the user that owns the schema of the routine. This is usually the creating user, although the database owner could be the creator as well.

For information about definer's and invoker's rights, see [CREATE FUNCTION](#) statement.

Each of these functions returns a string of up to 128 characters.

Syntax

```
CURRENT_USER
```

Example

```
splice> VALUES CURRENT_USER;
1
-----
SPICE
1 row selected
```

See Also

- » [USER function](#)
- » [SESSION_USER function](#)
- » [CREATE_FUNCTION statement](#)
- » [CREATE_PROCEDURE statement](#)

DATE

The DATE function returns a date from a value.

Syntax

```
DATE ( expression )
```

expression

An expression that can be any of the following:

- » A [LONG VARCHAR](#) value, which must represent a valid date in the form yyyyymm, where yyyy is a four-digit year value, and mm is a two-digit month value in the range 001 to 12.

Results

The returned result is governed by the following rules:

- » If the argument can be NULL, the result can be NULL; if the argument is NULL, the result is the NULL value.
- » If the argument is a date, timestamp, or valid string representation of a date or timestamp, the result is the date part of the value.
- » If the argument is a number, the result is the date that is n–1 days after January 1, 1970, where n is the integral part of the number.
- » If the argument is a string with a length of 7, the result is a string representation of the date.

Examples

This example results in an internal representation of ‘1988-12-25’.

```
splice> VALUES DATE('1988-12-25');
```

This example results in an internal representation of ‘1972-02-28’.

```
splice> VALUES DATE(789);
```

This example illustrates using date arithmetic with the DATE function:

```
splice> select Birthdate - DATE('11/22/1963') AS "DaysSinceJFK" FROM Players WHERE I  
D < 20;  
DaysSinceJ&  
-----  
8526  
8916  
9839  
8461  
9916  
6619  
6432  
7082  
7337  
7289  
9703  
5030  
9617  
6899  
9404  
7446  
7609  
9492  
9172  
  
19 rows selected
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE data type](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)
- » [TIME data type](#)

- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [Working with Dates](#) in the *Developer's Guide*

DAY

The DAY function returns the day part of a value.

Syntax

```
DAY ( expression )
```

expression

An expression that can be any of the following:

- » A `LONG VARCHAR` value.

Results

The returned result is an integer value in the range 1 to 31.

If the argument can be `NULL`, the result can be `NULL`; if the argument is `NULL`, the result is the `NULL` value.

Examples

Get the current date:

```
splice> VALUES(CURRENT_DATE);
1
-----
2015-10-25
1 row selected
```

Now get the current day only:

```
splice> VALUES(DAY(CURRENT_DATE));
1
-----
25
1 row selected
```

Get the day number for each player's birthdate:

```
splice> select Day(Birthdate) AS "Day-of-Birth"
      FROM Players
     WHERE ID < 20
    ORDER BY "Day-of-Birth";
Day-of-Bir&
-----
1
2
5
6
11
12
13
15
16
17
20
21
21
21
22
24
27
30
30

19 rows selected
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE data type](#)
- » [DATE function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)

- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [Working with Dates](#) in the *Developer's Guide*

DEGREES

The DEGREES function converts (approximately) a specified number from radians to degrees.

NOTE: The conversion from radians to degrees is not exact. You should not expect DEGREES(ACOS(0.5)) to return exactly 60.0.

Syntax

```
DEGREES ( number )
```

number

A [DOUBLE PRECISION](#) number that specifies the angle you want converted, in radians.

Example

```
splice> VALUES DEGREES(ACOS(0.5));
1
-----
60.00000000000001

1 row selected
```

Results

The data type of the returned value is a [DOUBLE PRECISION](#) number.

See Also

- » [DOUBLE PRECISION](#) data type
- » [ACOS](#) function
- » [ASIN](#) function
- » [ATAN](#) function
- » [ATAN2](#) function
- » [COS](#) function
- » [COSH](#) function

- » [COT function](#)
- » [RADIAN function](#)
- » [SIN function](#)
- » [SINH function](#)
- » [TAN function](#)
- » [TANH function](#)

DENSE_RANK()

`DENSE_RANK()` is a *ranking function* that returns the rank of a value within the ordered partition of values defined by its `OVER` clause. Ranking functions are a subset of window functions.

Syntax

```
DENSE_RANK() OVER ( overClause )
```

`overClause`

See the [OVER clause documentation](#).

NOTE: Ranking functions such as `DENSE_RANK` must include an `ORDER BY` clause in the `OVER` clause. This is because the ranking is calculated based on the ordering.

Results

The resulting data type is [BIGINT](#).

Usage

The `DENSE_RANK()` and [RANK\(\)](#) analytic functions are very similar. The difference shows up when there are multiple input rows that have the same ranking value. When that happens:

- » The `DENSE_RANK()` function always returns consecutive rankings: if values in the ranking column are the same, they receive the same rank, and the next number in the ranking sequence is then used to rank the row or rows that follow.
- » The `RANK()` function can generate non-consecutive ranking result values: if values in the ranking column are the same (tie values), they receive the same rank; however, the next number in the ranking sequence is then skipped, which means that `RANK` can return non-consecutive numbers.

Here's a simple example that shows the ranking produced by the two functions for input with duplicate values to illustrate that difference:

Value	RANK	DENSE_RANK
a	1	1
a	1	1
a	1	1
b	4	2
c	5	3
c	5	3
d	7	4
e	8	5

Example

The following query ranks the salaries of players, per team, whose salary is at least \$1 million.

```

SELECT DisplayName, Team, Season, Salary,
       DENSE_RANK() OVER (PARTITION BY Team ORDER BY Salary Desc) "RANK"
  FROM Players JOIN Salaries ON Salaries.ID=Players.ID
 WHERE Salary>999999 AND Season=2015;

```

DISPLAYNAME	TEAM	SEASON	SALARY	RANK
Mitch Hassleman	Cards	2015	17000000	1
Yuri Miletton	Cards	2015	15200000	2
James Grasser	Cards	2015	9375000	3
Jack Hellman	Cards	2015	8300000	4
Larry Lintos	Cards	2015	7000000	5
Jeremy Johnson	Cards	2015	4125000	6
Mitch Canepa	Cards	2015	3750000	7
Mitch Brandon	Cards	2015	3500000	8
Robert Cohen	Cards	2015	3000000	9
James Woegren	Cards	2015	2675000	10
Sam Culligan	Cards	2015	2652732	11
Barry Morse	Cards	2015	2379781	12
Michael Rastono	Cards	2015	2000000	13
Carl Vanamos	Cards	2015	2000000	13
Alex Wister	Cards	2015	1950000	14
Pablo Bonjourno	Cards	2015	1650000	15
Jonathan Pearlman	Cards	2015	1500000	16
Jan Bromley	Cards	2015	1200000	17
Martin Cassman	Giants	2015	20833333	1
Harry Pennello	Giants	2015	18500000	2
Tam Lassiter	Giants	2015	18000000	3
Buddy Painter	Giants	2015	17277777	4
Thomas Hillman	Giants	2015	12000000	5
Alex Paramour	Giants	2015	10250000	6
Jack Peepers	Giants	2015	9000000	7
Mark Briste	Giants	2015	8000000	8
Marcus Bamburger	Giants	2015	6950000	9
Jalen Ardson	Giants	2015	6000000	10
Steve Raster	Giants	2015	6000000	10
Sam Castleman	Giants	2015	5000000	11
Craig McGawn	Giants	2015	4800000	12
Norman Aikman	Giants	2015	4000000	13
Randy Varner	Giants	2015	4000000	13
Jason Lilliput	Giants	2015	4000000	13
Billy Bopper	Giants	2015	3600000	14
Greg Brown	Giants	2015	3600000	14
Mitch Lovell	Giants	2015	3578825	15
Bob Cranker	Giants	2015	3175000	16
Yuri Piamam	Giants	2015	2100000	17
Joseph Arkman	Giants	2015	1450000	18
Trevor Imhof	Giants	2015	1100000	19
Jason Minman	Giants	2015	1000000	20

42 rows selected

Here's the same query using RANK instead of DENSE_RANK. Note how tied rankings are handled differently:

```

SELECT DisplayName, Team, Season, Salary,
       RANK() OVER (PARTITION BY Team ORDER BY Salary Desc) "RANK"
  FROM Players JOIN Salaries ON Salaries.ID=Players.ID
 WHERE Salary>999999 AND Season=2015;

```

DISPLAYNAME	TEAM	SEASON	SALARY	RANK
Mitch Hassleman	Cards	2015	17000000	1
Yuri Milleton	Cards	2015	15200000	2
James Grasser	Cards	2015	9375000	3
Jack Hellman	Cards	2015	8300000	4
Larry Lintos	Cards	2015	7000000	5
Jeremy Johnson	Cards	2015	4125000	6
Mitch Canepa	Cards	2015	3750000	7
Mitch Brandon	Cards	2015	3500000	8
Robert Cohen	Cards	2015	3000000	9
James Woegren	Cards	2015	2675000	10
Sam Culligan	Cards	2015	2652732	11
Barry Morse	Cards	2015	2379781	12
Michael Rastono	Cards	2015	2000000	13
Carl Vanamos	Cards	2015	2000000	13
Alex Wister	Cards	2015	1950000	15
Pablo Bonjourno	Cards	2015	1650000	16
Jonathan Pearlman	Cards	2015	1500000	17
Jan Bromley	Cards	2015	1200000	18
Martin Cassman	Giants	2015	20833333	1
Harry Pennello	Giants	2015	18500000	2
Tam Lassiter	Giants	2015	18000000	3
Buddy Painter	Giants	2015	17277777	4
Thomas Hillman	Giants	2015	12000000	5
Alex Paramour	Giants	2015	10250000	6
Jack Peepers	Giants	2015	9000000	7
Mark Briste	Giants	2015	8000000	8
Marcus Bamburger	Giants	2015	6950000	9
Jalen Ardson	Giants	2015	6000000	10
Steve Raster	Giants	2015	6000000	10
Sam Castleman	Giants	2015	5000000	12
Craig McGawn	Giants	2015	4800000	13
Norman Aikman	Giants	2015	4000000	14
Randy Varner	Giants	2015	4000000	14
Jason Lilliput	Giants	2015	4000000	14
Billy Bopper	Giants	2015	3600000	17
Greg Brown	Giants	2015	3600000	17
Mitch Lovell	Giants	2015	3578825	19
Bob Cranker	Giants	2015	3175000	20
Yuri Piamam	Giants	2015	2100000	21
Joseph Arkman	Giants	2015	1450000	22
Trevor Imhof	Giants	2015	1100000	23
Jason Minman	Giants	2015	1000000	24

42 rows selected

See Also

- » Window and Aggregate functions
- » **BIGINT** data type
- » **RANK** function
- » **OVER** clause
- » [Using Window Functions](#) in the *Developer Guide*.

DOUBLE

The DOUBLE function returns a floating-point number corresponding to a:

- » number if the argument is a numeric expression
- » character string representation of a number if the argument is a string expression

Numeric to Double

```
DOUBLE [PRECISION] (NumericExpression )
```

NumericExpression

The argument is an expression that returns a value of any built-in numeric data type.

Results

The data type of the returned value is a `DOUBLE PRECISION` number.

If the argument can be `NULL`, the result can be `NULL`; if the argument is `NULL`, the result is the `NULL`value.

The result is the same value that would result if the argument were assigned to a double-precision floating-point column or variable.

Character String to Double

```
DOUBLE (StringExpression )
```

StringExpression

The argument can be of type `VARCHAR` in the form of a numeric constant. Leading and trailing blanks in argument are ignored.

Results

The data type of the returned value is a `DOUBLE PRECISION` number.

If the argument can be `NULL`, the result can be `NULL`; if the argument is `NULL`, the result is the `NULL`value.

The result is the same value that would result if the string was considered a constant and assigned to a double-precision floating-point column or variable.

Example

```
splice> VALUES DOUBLE(84.4);
1
-----
84.4
1 row selected
```

See Also

- » [About Data Types](#)

EXP

The EXP function returns e raised to the power of the specified number. The constant e is the base of the natural logarithms.

Syntax

```
EXP ( number )
```

number

A [DOUBLE PRECISION](#) number that specifies the exponent to which you want to raise e.

Example

```
splice> VALUES EXP(1.234);
1
-----
3.43494186080076

1 row selected
```

Results

The data type of the result is a [DOUBLE PRECISION](#) number.

See Also

- » [About Data Types](#)
- » [DOUBLE PRECISION](#) data type

EXTRACT

You can use the EXTRACT built-in function can use to extract specific information from date and time values.

Syntax

```
EXTRACT( infoType FROM dateExpr );
```

infoType

The value (information) that you want to extract and return from the date-time expression. This can be one of the following values:

YEAR

The four-digit year value is extracted from the date-time expression.

QUARTER

The single digit (1–4) quarter number is extracted from the date-time expression.

MONTH

The month number (1–12) is extracted from the date-time expression.

MONTHNAME

The full month name (e.g. September) is extracted from the date-time expression.

WEEK

The week-of-year number (1 is the first week) is extracted from the date-time expression.

WEEKDAY

The day-of-week number (1–7, with Monday as 1 and Sunday as 7) is extracted from the date-time expression.

WEEKDAYNAME

The day-of-week name (e.g. Tuesday) is extracted from the date-time expression.

DAYOFYEAR

The numeric day-of-year (0–366) is extracted from the date-time expression.

DAY

The numeric day-of-month (0–31) is extracted from the date-time expression.

HOUR

The numeric hour (0–23) is extracted from the date-time expression.

Note that Splice Machine `DATE` values do not include time information and will not work correctly with this *infoType*.

MINUTE

The numeric minute (0–59) is extracted from the date-time expression.

Note that Splice Machine `DATE` values do not include time information and will not work correctly with this *infoType*.

SECOND

The numeric second (0–59) is extracted from the date-time expression.

Note that Splice Machine `DATE` values do not include time information and will not work correctly with this *infoType*.

dateExpr

The date-time expression from which you wish to extract information.

Note that Splice Machine `DATE` values do not include time information and thus will not produce correct values if you specify HOUR, MINUTE, or SECOND infoTypes.

Examples

```

splice> SELECT Birthdate,
    EXTRACT (Quarter FROM Birthdate) "Quarter",
    EXTRACT (Week FROM Birthdate) "Week",
    EXTRACT(WeekDay FROM Birthdate) "Weekday"
  FROM Players
 WHERE ID < 20
 ORDER BY "Quarter";
BIRTHDATE |Quarter|Week|Weekday
-----
1987-03-27|1|13|5
1987-01-21|1|4|3
1991-01-15|1|3|2
1982-01-05|1|1|2
1990-03-22|1|12|4
1989-01-01|1|52|7
1988-04-20|2|16|3
1983-04-13|2|15|3
1990-06-16|2|24|6
1984-04-11|2|15|3
1981-07-02|3|27|4
1977-08-30|3|35|2
1989-08-21|3|34|1
1984-09-21|3|38|5
1990-10-30|4|44|2
1983-12-24|4|51|6
1983-11-06|4|44|7
1982-10-12|4|41|2
1989-11-17|4|46|5

19 rows selected

splice> values EXTRACT(monthname FROM '2009-09-02 11:22:33.04');
1
-----
September

splice> values EXTRACT(weekdayname FROM '2009-11-07 11:22:33.04');
1
-----
Saturday
1 row selected

splice> values EXTRACT(dayofyear FROM '2009-02-01 11:22:33.04');
1
-----
32
1 row selected

splice> values EXTRACT(hour FROM '2009-07-02 11:22:33.04');
1

```

```
-----
11
1 row selected

splice> values EXTRACT(minute FROM '2009-07-02 11:22:33.04');
1
-----
22
1 row selected

splice> values EXTRACT(second FROM '2009-07-02 11:22:33.04');
1
-----
33

1 row selected
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)
- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [Working with Dates in the Developer's Guide](#)

FIRST_VALUE

`FIRST_VALUE` is a window function that returns the values of a specified expression that is evaluated at the first row of a window for the current row. This means that you can select a first value from a set of rows without having to use a self join.

Syntax

```
FIRST_VALUE ( expression [ {IGNORE | RESPECT} NULLS ] ) OVER ( overClause )
```

expression

The expression to evaluate; typically a column name or computation involving a column name.

IGNORE NULLS

If this optional qualifier is specified, `NULL` values are ignored, and the first non-`NULL` value is evaluated.

If you specify this and all values are `NULL`, `FIRST_VALUE` returns `NULL`.

RESPECT NULLS

This qualifier is the default behavior: it specifies that the first value is always returned, even if it is `NULL`.

overClause

See the [OVER clause documentation](#).

Usage Notes

Splice Machine recommends that you use the `FIRST_VALUE` function with the [ORDER BY](#) clause to produce deterministic results.

Results

Returns value(s) resulting from the evaluation of the specified expression; the return type is of the same value type as the data stored in the column used in the expression..

- » `FIRST_VALUE` returns the first value in the set, unless that value is `NULL` and you have specified the `IGNORE NULLS` qualifier; if you've specified `IGNORE NULLS`, this function returns the first non-`NULL` value in the set.
- » If all values in the set are `NULL`, `FIRST_VALUE` always returns `NULL`.

NOTE: Splice Machine always sorts `NULL` values first in the results.

Examples

The following query finds all players with 10 or more HomeRuns, and compares each player's home run count with the lowest total within that group on his team:

```
splice> SELECT Team, DisplayName, HomeRuns,
    FIRST_VALUE(HomeRuns) OVER (PARTITION BY Team ORDER BY HomeRuns
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) "Least"
    FROM Players JOIN Batting ON Players.ID=Batting.ID
    WHERE HomeRuns > 10
    ORDER BY Team, HomeRuns DESC;
TEAM      | DISPLAYNAME          | HOMER& | Least
-----|-----|-----|-----|-----|
Cards    | Mitch Canepa         | 28     | 11
Cards    | Jonathan Pearlman   | 17     | 11
Cards    | Roger Green           | 17     | 11
Cards    | Michael Rastono      | 13     | 11
Cards    | Jack Hellman          | 13     | 11
Cards    | Kelly Wacherman       | 11     | 11
Giants   | Bob Cranker           | 21     | 12
Giants   | Buddy Painter          | 19     | 12
Giants   | Billy Bopper           | 18     | 12
Giants   | Mitch Duffer           | 12     | 12
10 rows selected
```

See Also

- » [Window and Aggregate functions](#)
- » [AVG function](#)
- » [COUNT function](#)
- » [LAG function](#)
- » [LAST_VALUE function](#)
- » [LEAD function](#)
- » [MIN function](#)
- » [SUM function](#)
- » [OVER clause](#)
- » [Using Window Functions](#)

FLOOR

The FLOOR function rounds the specified number down, and returns the largest number that is less than or equal to the specified number.

Syntax

```
FLOOR ( number )
```

number

A [DOUBLE PRECISION](#) number.

Example

```
splice> VALUES FLOOR(84.4);
1
-----
84
1 row selected
```

Results

The data type of the result is a [DOUBLE PRECISION](#) number. The returned value is equal to a mathematical integer.

- » If the specified number is `NULL`, the result of this function is `NULL`.
- » If the specified number is equal to a mathematical integer, the result of this function is the same as the specified number.
- » If the specified number is zero (0), the result of this function is zero.

See Also

- » [About Data Types](#)
- » [DOUBLE PRECISION](#) data type

HOUR

The HOUR function returns the hour part of a value.

Syntax

```
HOUR ( expression )
```

expression

An expression that can be a time, timestamp, or a valid character string representation of a time or timestamp.

Results

The returned result is an integer value in the range 0 to 24.

If the argument can be NULL, the result can be NULL; if the argument is NULL, the result is the NULLvalue.

Example

```
splice> values ( NOW, HOUR(NOW), MINUTE(NOW), SECOND(NOW) );
1 | 2 | 3 | 4
-----
2015-11-12 17:48:55.217 | 17 | 48 | 55.217
1 row selected
```

See Also

- » [About Data Types](#)
- » [TIME data value](#)
- » [TIMESTAMP data value](#)
- » [MINUTE function](#)
- » [TIMESTAMP function](#)
- » [TIMESTAMPADD function](#)
- » [TIMESTAMPDIFF function](#)

INITCAP

The INITCAP function converts the first letter of each word in a string to uppercase, and converts any remaining characters in each word to lowercase. Words are delimited by white space characters, or by characters that are not alphanumeric.

Syntax

```
INITCAP( charExpression );
```

charExpression

The string to be converted. This can be a CHAR or VARCHAR data type, or another type that gets implicitly converted.

Results

The returned string has the same data type as the input *charExpression*.

Examples

```
splice> VALUES( INITCAP('this is a test') );
1
```

```
-----  
This Is A Test  
1 row selected
```

```
splice> VALUES( INITCAP('tHIS is a test') );
1
```

```
-----  
This Is A Test  
1 row selected
```

See Also

- » [About Data Types](#)
- » [Concatenation operator](#)
- » [INSTR function](#)
- » [LCASE function](#)
- » [LENGTH function](#)

- » [LOCATE function](#)
- » [LTRIM function](#)
- » [REGEX_LIKE operator](#)
- » [REPLACE function](#)
- » [RTRIM function](#)
- » [SUBSTR function](#)
- » [TRIM function](#)
- » [UCASE function](#)

INSTR

The INSTR function returns the index of the first occurrence of a substring in a string.

Syntax

```
INSTR(str, substring)
```

str

The string in which to search for the substring.

substring

The substring to search for.

Results

Returns the index in *str* of the first occurrence of *substring*.

The first index is 1.

If *substring* is not found, INSTR returns 0.

Examples

```
splice> SELECT DisplayName, INSTR(DisplayName, 'Pa') "Position"
      FROM Players
     WHERE (INSTR(DisplayName, 'Pa') > 0)
      ORDER BY DisplayName;
DISPLAYNAME          | Position
-----
Alex Paramour       | 6
Buddy Painter      | 7
Jeremy Packman     | 8
Pablo Bonjourno    | 1
Paul Kaster         | 1
5 rows selected
```

See Also

» About Data Types

- » Concatenation operator
- » [INITCAP](#) function
- » [LCASE](#) function
- » [LENGTH](#) function
- » [LOCATE](#) function
- » [LTRIM](#) function
- » [REGEX_LIKE](#) operator
- » [REPLACE](#) function
- » [RTRIM](#) function
- » [SUBSTR](#) function
- » [TRIM](#) function
- » [UCASE](#) function

INTEGER

The INTEGER function returns an integer representation of a number or character string in the form of an integer constant.

Syntax

```
INT[TEGER] (NumericExpression | CharacterExpression )
```

NumericExpression

An expression that returns a value of any built-in numeric data type.

CharacterExpression

An expression that returns a character string value of length not greater than the maximum length of a character constant. Leading and trailing blanks are eliminated and the resulting string must conform to the rules for forming an SQL integer constant. The character string cannot be a long string.

Results

The result of the function is a large integer.

- » If the argument can be NULL, the result can be NULL; if the argument is NULL, the result is the NULLvalue.
- » If the argument is a numeric-expression, the result is the same number that would occur if the argument were assigned to a large integer column or variable. If the whole part of the argument is not within the range of integers, an error occurs. The decimal part of the argument is truncated if present.
- » If the argument is a character-expression, the result is the same number that would occur if the corresponding integer constant were assigned to a large integer column or variable.

Example

The following query truncates the number of innings pitches by using the INTEGER function:

```
splice> SELECT DisplayName, INTEGER(Innings) "Innings"
      FROM Pitching JOIN Players ON Pitching.ID=Players.ID
     WHERE Innings > 50
     ORDER BY Innings DESC;
DISPLAYNAME          | Innings
-----
Marcus Bamburger    | 218
Jason Larrimore     | 218
Milt Warrimore       | 181
Carl Marin           | 179
Charles Heillman     | 177
Larry Lintos          | 175
Randy Varner          | 135
James Grasser          | 129
Thomas Hillman         | 123
Jack Peepers          | 110
Tam Lassiter          | 76
Yuri Piamam           | 76
Ken Straiter          | 74
Gary Kosovo            | 73
Tom Rather             | 68
Steve Mossely          | 63
Carl Vanamos           | 61
Martin Cassman         | 60
Tim Lentleson          | 60
Sam Castleman          | 58
Steve Raster            | 57
Mitch Lovell            | 55
Harold Sermer           | 51

23 rows selected
```

See Also

- » [About Data Types](#)

LAG

LAG returns the values of a specified expression that is evaluated at the specified offset number of rows before the current row in a window.

Syntax

```
LAG ( expression [ , offset ] ) OVER ( overClause )
```

expression

The expression to evaluate; typically a column name or computation involving a column name.

offset

An integer value that specifies the offset (number of rows) from the current row at which you want the expression evaluated.

The default value is 1.

overClause

See the [OVER clause documentation](#).

Our current implementation of this function does not allow for specifying a default value, as is possible in some other database software.

Usage Notes

Splice Machine recommends that you use the LAG function with the [ORDER BY](#) clause to produce deterministic results.

Results

Returns value(s) resulting from the evaluation of the specified expression; the return type is of the same value type as the date stored in the column used in the expression.

Examples

The following example shows the salaries per position for players in our baseball database, grouped by position, and ordered from highest salary to lowest for each position:

```
splice> SELECT Position, Players.ID, Salary,
    LAG(Salary) OVER (PARTITION BY Position ORDER BY Salary DESC) "PrevHigherSalary"
    FROM Players JOIN Salaries ON Players.ID=Salaries.ID
    WHERE Salary > 999999
    ORDER BY Position, Salary DESC;
POS& | ID      | SALARY          | PrevHigherSalary
-----+
1B   | 2       | 3600000        | NULL
1B   | 63      | 2379781        | 3600000
1B   | 50       | 2000000        | 2379781
3B   | 14      | 4800000        | NULL
3B   | 53       | 3750000        | 4800000
C    | 1       | 17277777       | NULL
C    | 49      | 15200000       | 17277777
CF   | 7       | 10250000       | NULL
CF   | 59      | 4125000        | 10250000
CF   | 55      | 1650000        | 4125000
LF   | 54      | 17000000       | 2000000
LF   | 6       | 4000000        | 17000000
LF   | 27      | 1100000        | 4000000
P    | 34      | 20833333       | NULL
P    | 33      | 18000000       | 20833333
P    | 31      | 12000000       | 18000000
P    | 76      | 9375000        | 12000000
P    | 32      | 9000000        | 9375000
P    | 75      | 7000000        | 9000000
P    | 28      | 6950000        | 7000000
P    | 40      | 6000000        | 6950000
P    | 41      | 6000000        | 6000000
P    | 46      | 5000000        | 6000000
P    | 30      | 4000000        | 5000000
P    | 43      | 4000000        | 4000000
P    | 35      | 3578825        | 4000000
P    | 86      | 3500000        | 3578825
P    | 82      | 3000000        | 3500000
P    | 88      | 2675000        | 3000000
P    | 90      | 2652732        | 2675000
P    | 36      | 2100000        | 2652732
P    | 79      | 2000000        | 2100000
P    | 80      | 1950000        | 2000000
P    | 94      | 1200000        | 1950000
RF   | 8       | 18500000       | NULL
RF   | 56      | 8300000        | 18500000
RF   | 12      | 8000000        | 8300000
RF   | 10      | 1000000        | 8000000
SS   | 4       | 3175000        | NULL
SS   | 52      | 1500000        | 3175000
UT   | 17      | 1450000        | NULL
```

41 rows selected

See Also

- » Window and Aggregate functions
- » [AVG function](#)
- » [COUNT function](#)
- » [FIRST_VALUE function](#)
- » [LAST_VALUE function](#)
- » [LEAD function](#)
- » [MIN function](#)
- » [SUM function](#)
- » [OVER clause](#)
- » [Using Window Functions](#) in the *Developer Guide*.

LAST_DAY

The `LAST_DAY` function returns the date of the last day of the month that contains the input date.

Syntax

```
LAST_DAY ( dateExpression )
```

dateExpression

A date value.

Results

The return type is always `DATE`, regardless of the data type of the *dateExpression*.

Examples

Examples:

```
splice> values (LAST_DAY(CURRENT_DATE));
1
```

```
-----  
2015-11-30
```

```
splice> values (LAST_DAY(DATE(CURRENT_TIMESTAMP)));
1
```

```
-----  
2015-11-30
```

```
splice> SELECT DISPLAYNAME, BirthDate, LAST_DAY(BirthDate) "MonthEnd"
      FROM Players
     WHERE MONTH(BirthDate) IN (2, 5, 12);
```

DISPLAYNAME	BIRTHDATE	MonthEnd
Tam Croonster	1980-12-19	1980-12-31
Jack Peepers	1981-05-31	1981-05-31
Jason Martell	1982-02-01	1982-02-28
Kameron Fannais	1982-05-24	1982-05-31
Jonathan Pearlman	1982-05-28	1982-05-31
Greg Brown	1983-12-24	1983-12-31
Edward Erdman	1985-12-21	1985-12-31
Jonathan Wilson	1986-05-14	1986-05-31
Reed Lister	1986-12-16	1986-12-31
Larry Lintos	1987-05-12	1987-05-31
Taylor Trantula	1987-12-17	1987-12-31
Tim Lentleson	1988-02-21	1988-02-29
Cameron Silliman	1988-12-21	1988-12-31
Nathan Nickels	1989-05-04	1989-05-31
Tom Rather	1990-05-29	1990-05-31
Mo Grandosi	1992-02-16	1992-02-29

```
16 rows selected
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [MONTH function](#)

- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)
- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [*Working with Dates* in the *Developer's Guide*](#)

LAST_VALUE

`LAST_VALUE` is a window function that returns the values of a specified expression that is evaluated at the last row of a window for the current row. This means that you can select a last value from a set of rows without having to use a self join.

Syntax

```
LAST_VALUE ( expression [ {IGNORE | RESPECT} NULLS ] ) OVER ( overClause )
```

expression

The expression to evaluate; typically a column name or computation involving a column name.

IGNORE NULLS

If this optional qualifier is specified, `NULL` values are ignored, and the first non-`NULL` value is evaluated.

If you specify this and all values are `NULL`, `LAST_VALUE` returns `NULL`.

RESPECT NULLS

This qualifier is the default behavior: it specifies that the last value is always returned, even if it is `NULL`.

overClause

See the [OVER clause documentation](#).

Usage Notes

Splice Machine recommends that you use the `LAST_VALUE` function with the `ORDER BY` clause to produce deterministic results.

Results

Returns value(s) resulting from the evaluation of the specified expression; the return type is of the same value type as the date stored in the column used in the expression..

- » `LAST_VALUE` returns the last value in the set, unless that value is `NULL` and you have specified the `IGNORE NULLS` qualifier; if you've specified `IGNORE NULLS`, this function returns the last non-`NULL` value in the set.
- » If all values in the set are `NULL`, `LAST_VALUE` always returns `NULL`.

NOTE: Splice Machine always sorts `NULL` values first in the results.

Examples

The following query finds all players with 10 or more HomeRuns, and compares each player's home run count with the highest total on his team:

```
splice> SELECT Team, DisplayName, HomeRuns,
    LAST_VALUE(HomeRuns) OVER (PARTITION BY Team ORDER BY HomeRuns
    ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) "Most"
    FROM Players JOIN Batting ON Players.ID=Batting.ID
    WHERE HomeRuns > 10
    ORDER BY Team, HomeRuns DESC;
TEAM      | DISPLAYNAME          | HOMER& | Most
-----+-----+-----+-----+
Cards    | Mitch Canepa        | 28     | 28
Cards    | Jonathan Pearlman   | 17     | 28
Cards    | Roger Green          | 17     | 28
Cards    | Michael Rastono      | 13     | 28
Cards    | Jack Hellman         | 13     | 28
Cards    | Kelly Wacherman      | 11     | 28
Giants   | Bob Cranker          | 21     | 21
Giants   | Buddy Painter         | 19     | 21
Giants   | Billy Bopper          | 18     | 21
Giants   | Mitch Duffer          | 12     | 21
-----+-----+-----+-----+
10 rows selected
```

See Also

- » [Window and Aggregate functions](#)
- » [AVG function](#)
- » [COUNT function](#)
- » [FIRST_VALUE function](#)
- » [LAG function](#)
- » [LEAD function](#)
- » [MIN function](#)
- » [SUM function](#)
- » [OVER clause](#)
- » [Using Window Functions](#) in the *Developer Guide*.

LCASE or LOWER

LCASE or LOWER returns a string in which all alphabetic characters in the input character expression have been converted to lowercase.

NOTE: LOWER and LCASE follow the database locale.

Syntax

```
LCASE or LOWER ( CharacterExpression )
```

CharacterExpression

A [LONG VARCHAR](#) data type, or any built-in type that is implicitly converted to a string (but not a bit expression).

Results

The data type of the result is as follows:

- » If the *CharacterExpression* evaluates to NULL, this function returns NULL.
- » If the *CharacterExpression* is of type [CHAR](#).
- » If the *CharacterExpression* is of type [LONG VARCHAR](#).
- » Otherwise, the return type is [VARCHAR](#).

The length and maximum length of the returned value are the same as the length and maximum length of the parameter.

Examples

```
splice> SELECT LCASE(DisplayName)
   FROM Players
  WHERE ID < 11;
```

```
1
```

```
-----  
buddy painter  
billy bopper  
john purser  
bob cranker  
mitch duffer  
norman aikman  
alex paramour  
harry pennello  
greg brown  
jason minman
```

```
10 rows selected
```

See Also

- » [About Data Types](#)
- » [Concatenation operator](#)
- » [INITCAP function](#)
- » [INSTR function](#)
- » [LENGTH function](#)
- » [LOCATE function](#)
- » [LTRIM function](#)
- » [REGEX_LIKE operator](#)
- » [REPLACE function](#)
- » [RTRIM function](#)
- » [SUBSTR function](#)
- » [TRIM function](#)
- » [UCASE function](#)

LEAD

LEAD is a window function that returns the values of a specified expression that is evaluated at the specified offset number of rows after the current row in a window.

Syntax

```
LEAD ( expression [ , offset ] ) OVER ( overClause )
```

expression

The expression to evaluate; typically a column name or computation involving a column name.

offset

An integer value that specifies the offset (number of rows) from the current row at which you want the expression evaluated.

The default value is 1.

overClause

See the [OVER clause documentation](#).

Our current implementation of this function does not allow for specifying a default value, as is possible in some other database software.

Usage Notes

Splice Machine recommends that you use the LEAD function with the [ORDER BY](#) clause to produce deterministic results.

Results

Returns value(s) resulting from the evaluation of the specified expression; the return type is of the same value type as the data stored in the column used in the expression.

Examples

```
splice> SELECT DisplayName, Position, Salary,
    LEAD(SALARY) OVER (PARTITION BY Position ORDER BY Salary DESC) "NextLowerSalary"
    FROM Players JOIN Salaries ON Players.ID=Salaries.ID
    WHERE Salary>999999
    ORDER BY Position, Salary DESC;
```

DISPLAYNAME	POS&	SALARY	NextLowerSalary
Billy Bopper	1B	3600000	2379781
Barry Morse	1B	2379781	2000000
Michael Rastono	1B	2000000	NULL
Craig McGawn	3B	4800000	3750000
Mitch Canepa	3B	3750000	NULL
Buddy Painter	C	17277777	15200000
Yuri Milleton	C	15200000	NULL
Alex Paramour	CF	10250000	4125000
Jeremy Johnson	CF	4125000	1650000
Pablo Bonjourno	CF	1650000	NULL
Mitch Hassleman	LF	17000000	4000000
Norman Aikman	LF	4000000	1100000
Trevor Imhof	LF	1100000	NULL
Greg Brown	OF	3600000	NULL
Martin Cassman	P	20833333	18000000
Tam Lassiter	P	18000000	12000000
Thomas Hillman	P	12000000	9375000
James Grasser	P	9375000	9000000
Jack Peepers	P	9000000	7000000
Larry Lintos	P	7000000	6950000
Marcus Bamburger	P	6950000	6000000
Jalen Ardson	P	6000000	6000000
Steve Raster	P	6000000	5000000
Sam Castleman	P	5000000	4000000
Randy Varner	P	4000000	4000000
Jason Lilliput	P	4000000	3578825
Mitch Lovell	P	3578825	3500000
Mitch Brandon	P	3500000	3000000
Robert Cohen	P	3000000	2675000
James Woegren	P	2675000	2652732
Sam Culligan	P	2652732	2100000
Yuri Piamam	P	2100000	2000000
Carl Vanamos	P	2000000	1950000
Alex Wister	P	1950000	1200000
Jan Bromley	P	1200000	NULL
Harry Pennello	RF	18500000	8300000
Jack Hellman	RF	8300000	8000000
Mark Briste	RF	8000000	1000000
Jason Minman	RF	1000000	NULL
Bob Cranker	SS	3175000	1500000
Jonathan Pearlman	SS	1500000	NULL
Joseph Arkman	UT	1450000	NULL

42 rows selected

See Also

- » Window and Aggregate functions
- » [AVG function](#)
- » [COUNT function](#)
- » [FIRST_VALUE function](#)
- » [LAG function](#)
- » [LAST_VALUE function](#)
- » [MIN function](#)
- » [SUM function](#)
- » [OVER clause](#)
- » [Using Window Functions](#) in the *Developer Guide*.

LENGTH

The LENGTH function returns the number of characters in a character string expression or bit string expression.

NOTE: Since all built-in data types are implicitly converted to strings, this function can act on all built-in data types.

Syntax

```
LENGTH ( { CharacterExpression | BitExpression } )
```

CharacterExpression

A character string expression.

BitExpression

A bit string expression.

Results

The result data type is an integer value.

Examples

The following three examples show the values returned by the LENGTH function for string, integer, and bit string values.

```
splice> SELECT DisplayName, LENGTH(DisplayName) "NameLen"
   FROM Players
  WHERE ID < 11
 ORDER BY "NameLen";
```

DISPLAYNAME	NameLen
Greg Brown	10
John Purser	11
Bob Cranker	11
Billy Bopper	12
Mitch Duffer	12
Jason Minman	12
Buddy Painter	13
Norman Aikman	13
Alex Paramour	13
Harry Pennello	14

10 rows selected

```
splice> SELECT ID,
    LENGTH(CAST(ID AS SMALLINT)) "SMALLINT",
    LENGTH(CAST(ID AS INT)) "INT",
    LENGTH(CAST(ID AS BIGINT)) "BIGINT",
    LENGTH(CAST(ID AS DECIMAL)) "DECIMAL5",
    LENGTH(CAST(ID AS DECIMAL(15,10))) "DECIMAL15",
    LENGTH(CAST(ID AS DECIMAL(30,25))) "DECIMAL30"
   FROM Players
  WHERE ID<11;
```

ID	SMALLINT	INT	BIGINT	DECIMAL5	DECIMAL15	DECIMAL30
1	2	4	8	3	8	16
2	2	4	8	3	8	16
3	2	4	8	3	8	16
4	2	4	8	3	8	16
5	2	4	8	3	8	16
6	2	4	8	3	8	16
7	2	4	8	3	8	16
8	2	4	8	3	8	16
9	2	4	8	3	8	16
10	2	4	8	3	8	16

10 rows selected

```
splice> VALUES LENGTH(X'FF'),
    LENGTH(X'FFFF'),
    LENGTH(X'FFFFFFF'),
    LENGTH(X'FFFFFFFFFFFFFF');
```

1

```
2  
4  
8  
  
4 rows selected
```

See Also

- » [About Data Types](#)
- » [Concatenation operator](#)
- » [INITCAP function](#)
- » [INSTR function](#)
- » [LCASE function](#)
- » [LOCATE function](#)
- » [LTRIM function](#)
- » [REGEX_LIKE operator](#)
- » [REPLACE function](#)
- » [RTRIM function](#)
- » [SUBSTR function](#)
- » [TRIM function](#)
- » [UCASE function](#)

LN or LOG

The `LN` and `LOG` functions return the natural logarithm (base e) of the specified number.

Syntax

```
LN ( number )
LOG ( number )
```

number

A `DOUBLE PRECISION` number that is greater than zero (0).

Example

```
splice> VALUES( LOG(84.4), LN(84.4) );
1          | 2
-----
4.435674016019115 | 4.435674016019115
1 row selected
```

Results

The data type of the returned value is a `DOUBLE PRECISION` number.

- » If the specified number is `NULL`, the result of these functions is `NULL`.
- » If the specified number is zero or a negative number, an exception is returned that indicates that the value is out of range (SQL state 22003).

See Also

- » [About Data Types](#)
- » [DOUBLE PRECISION data type](#)

LOCATE

The LOCATE function is used to search for a string (the *needle*) within another string (the *haystack*). If the desired string is found, LOCATE returns the index at which it is found. If the desired string is not found, LOCATE returns 0.

Syntax

```
LOCATE ( CharacterExpression1, CharacterExpression2 [, StartPosition] )
```

CharacterExpression1

A character expression that specifies the string to search **for** in *CharacterExpression2*, sometimes called the needle.

CharacterExpression2

A character expression that specifies the string in which to search, sometimes called the haystack.

StartPosition

(Optional). Specifies the position in *CharacterExpression2* at which the search is to start. This defaults to the start of *CharacterExpression2*, which is the value 1.

Results

The return type for LOCATE is an integer that indicates the index position within the second argument at which the first argument was first located. Index positions start with 1.

- » If the first argument is not found in the second argument, LOCATE returns 0.
- » If the first argument is an empty string (' '), LOCATE returns the value of the third argument (or 1 if it was not provided), even if the second argument is also an empty string.
- » If a NULL value is passed for either of the CharacterExpression arguments, NULL is returned

Examples

```
splice> SELECT DisplayName, LOCATE('Pa', DisplayName, 3) "Position"
   FROM Players
  WHERE (INSTR(DisplayName, 'Pa') > 0)
 ORDER BY DisplayName;
DISPLAYNAME          | Position
-----
Alex Paramour       | 6
Buddy Painter      | 7
Jeremy Packman     | 8
Pablo Bonjourno    | 0
Paul Kaster         | 0

5 rows selected
```

See Also

- » [About Data Types](#)
- » [Concatenation operator](#)
- » [INITCAP function](#)
- » [INSTR function](#)
- » [LCASE function](#)
- » [LOCATE function](#)
- » [LTRIM function](#)
- » [REGEX_LIKE operator](#)
- » [REPLACE function](#)
- » [RTRIM function](#)
- » [SUBSTR function](#)
- » [TRIM function](#)
- » [UCASE function](#)

LOG10

The LOG10 function returns the base-10 logarithm of the specified number.

Syntax

```
LOG10 ( number )
```

number

A [DOUBLE PRECISION](#) number that is greater than zero (0).

Results

The data type of the returned value is a [DOUBLE PRECISION](#) number.

- » If the specified number is NULL, the result of this function is NULL.
- » If the specified number is zero or a negative number, an exception is returned that indicates that the value is out of range (SQL state 22003).

Example

```
splice> VALUES LOG10(84.4);
1
-----
1.926342446625655
1 row selected
```

See Also

- » [About Data Types](#)
- » [DOUBLE PRECISION](#) data type

LTRIM

LTRIM removes blanks from the beginning of a character string expression.

Syntax

```
LTRIM(CharacterExpression)
```

CharacterExpression

A `LONG VARCHAR` data type, or any built-in type that is implicitly converted to a string.

Results

A character string expression. If the *CharacterExpression* evaluates to `NULL`, this function returns `NULL`.

Example

```
splice> VALUES LTRIM('      Space Case      ');
1
-----
Space Case          --- This is the string 'Space Case      '
```

See Also

- » [About Data Types](#)
- » [Concatenation operator](#)
- » [INITCAP function](#)
- » [INSTR function](#)
- » [LCASE function](#)
- » [LENGTH function](#)
- » [LOCATE function](#)
- » [REGEX LIKE operator](#)
- » [REPLACE function](#)
- » [RTRIM function](#)
- » [SUBSTR function](#)

» [TRIM function](#)

» [UCASE function](#)

MAX

MAX evaluates the maximum of an expression over a set of rows. You can use it as a window (analytic) function.

Syntax

```
MAX ( [ DISTINCT | ALL ] Expression )
```

DISTINCT

If this qualifier is specified, duplicates are eliminated.

ALL

If this qualifier is specified, all duplicates are retained. This is the default value.

Expression

An expression that evaluates to a numeric data type: [SMALLINT](#).

The expression can contain multiple column references or expressions, but it cannot contain another aggregate or subquery, and it must evaluate to an ANSI SQL numeric data type. This means that you can call methods that evaluate to ANSI SQL data types.

If an expression evaluates to NULL, the aggregate skips that value.

Usage

Only one DISTINCT aggregate expression per *Expression* is allowed. For example, the following query is not valid:

```
-- Not a valid query:  
SELECT COUNT(DISTINCT flying_time),  
       MAX (DISTINCT miles)  
  FROM Flights;
```

NOTE: Since duplicate values do not change the computation of the maximum value, the DISTINCT and ALL qualifiers have no impact on this function.

The *Expression* can contain multiple column references or expressions, but it cannot contain another aggregate or subquery. It must evaluate to a built-in data type. You can therefore call methods that evaluate to built-in data types. (For example, a method that returns a *java.lang.Integer* or *int* evaluates to an INTEGER.) If an expression evaluates to NULL, the aggregate skips that value.

Results

The resulting data type is the same as the expression on which it operates; it will never overflow.

The comparison rules for the *Expression*'s type determine the resulting maximum value. For example, if you supply a `VARCHAR` argument, the number of blank spaces at the end of the value can affect how the maximum value is evaluated: if the values '`z`' and '`z`' are both stored in a column, you cannot control which one will be returned as the maximum, because blank spaces are ignored for character comparisons.

Examples

This example finds the birthdate of the youngest player in our database:

```
splice> SELECT MAX (BirthDate) FROM Players;
1
-----
1992-10-19
```

This example finds the maximum number of singles, doubles, triples and homeruns by any player in the database:

```
splice> SELECT MAX(Singles) "Singles", MAX(DOUBLES) "Doubles",
      MAX(Triples) "Triples", Max(HomeRuns) "HomeRuns"
      FROM Batting;
Singl&|Doubl&|Tripl&|HomeR&
-----
130 | 44 | 7 | 28
1 row selected
```

Analytic Example

The following shows the homeruns hit by all batters who hit more than 10, compared to the most Homeruns by a player who hit 10 or more on his team:

```
splice> SELECT Team, DisplayName, HomeRuns,
    MAX(HomeRuns) OVER (PARTITION BY Team ORDER BY HomeRuns
    ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) "Most"
    FROM Players JOIN Batting ON Players.ID=Batting.ID
    WHERE HomeRuns > 10
    ORDER BY Team, HomeRuns DESC;
TEAM      |DISPLAYNAME          | HOMER& | Most
-----|-----|-----|-----|
Cards    |Mitch Canepa        | 28     | 28
Cards    |Jonathan Pearlman   | 17     | 28
Cards    |Roger Green           | 17     | 28
Cards    |Michael Rastono       | 13     | 28
Cards    |Jack Hellman          | 13     | 28
Cards    |Kelly Wacherman        | 11     | 28
Giants   |Bob Cranker           | 21     | 21
Giants   |Buddy Painter          | 19     | 21
Giants   |Billy Bopper            | 18     | 21
Giants   |Mitch Duffer           | 12     | 21
10 rows selected
```

See Also

- » [About Data Types](#)
- » [Window and Aggregate Functions](#)
- » [AVG function](#)
- » [COUNT function](#)
- » [MIN function](#)
- » [SUM function](#)
- » [OVER clause](#)

MIN

MIN evaluates the minimum of an expression over a set of rows. You can use it as an window (analytic) function.

Syntax

```
MIN ( [ DISTINCT | ALL ] Expression )
```

DISTINCT

If this qualifier is specified, duplicates are eliminated.

ALL

If this qualifier is specified, all duplicates are retained. This is the default value.

Expression

An expression that evaluates to a numeric data type: [SMALLINT](#).

The expression can contain multiple column references or expressions, but it cannot contain another aggregate or subquery, and it must evaluate to an ANSI SQL numeric data type. This means that you can call methods that evaluate to ANSI SQL data types.

If an expression evaluates to `NULL`, the aggregate skips that value.

Usage

Only one DISTINCT aggregate expression per *Expression* is allowed. For example, the following query is not valid:

```
--- Not a valid query:  
SELECT COUNT (DISTINCT flying_time),  
       MIN (DISTINCT miles)  
  FROM Flights;
```

NOTE: Since duplicate values do not change the computation of the minimum value, the DISTINCT and ALL qualifiers have no impact on this function.

The *Expression* can contain multiple column references or expressions, but it cannot contain another aggregate or subquery. It must evaluate to a built-in data type. You can therefore call methods that evaluate to built-in data types. (For example, a method that returns a `java.lang.Integer` or `int` evaluates to an `INTEGER`.) If an expression evaluates to `NULL`, the aggregate skips that value.

Results

The resulting data type is the same as the expression on which it operates; it will never overflow.

The comparison rules for the *Expression*'s type determine the resulting minimum value. For example, if you supply a `VARCHAR` argument, the number of blank spaces at the end of the value can affect how the minimum value is evaluated: if the values '`z`' and ' `z`' are both stored in a column, you cannot control which one will be returned as the minimum, because blank spaces are ignored for character comparisons.

Examples

```
splice> SELECT MIN (BirthDate) FROM Players;
1
-----
1975-07-13
```

This example finds the minimum number of walks and strikeouts by any pitcher in the database:

```
splice> SELECT MIN(Walks) "Walks", Min(Strikeouts) "Strikeouts"
   FROM Pitching JOIN Players on Pitching.ID=Players.ID
   WHERE Position='P';
Walks |Strikeouts
-----
1    |1
1 row selected
```

Analytic Example

The following shows the homeruns hit by all batters who hit more than 10, compared to the least number of Homeruns by a player who hit 10 or more on his team:

```
splice> SELECT Team, DisplayName, HomeRuns,
    MIN(HomeRuns) OVER (PARTITION BY Team ORDER BY HomeRuns
    ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) "Least"
    FROM Players JOIN Batting ON Players.ID=Batting.ID
    WHERE HomeRuns > 10
    ORDER BY Team, HomeRuns DESC;
TEAM      |DISPLAYNAME          | HOMER& |Least
-----|-----|-----|-----|-----|
Cards    |Mitch Canepa        | 28     | 11
Cards    |Jonathan Pearlman   | 17     | 11
Cards    |Roger Green           | 17     | 11
Cards    |Michael Rastono       | 13     | 11
Cards    |Jack Hellman          | 13     | 11
Cards    |Kelly Wacherman        | 11     | 11
Giants   |Bob Cranker           | 21     | 12
Giants   |Buddy Painter          | 19     | 12
Giants   |Billy Bopper            | 18     | 12
Giants   |Mitch Duffer           | 12     | 12
10 rows selected
```

See Also

- » [Window and Aggregate functions](#)
- » [About Data Types](#)
- » [AVG function](#)
- » [COUNT function](#)
- » [MAX function](#)
- » [SUM function](#)
- » [OVER clause](#)

MINUTE

The MINUTE function returns the minute part of a value.

Syntax

```
MINUTE ( expression )
```

expression

An expression that can be a time, timestamp, or a valid character string representation of a time or timestamp.

Results

The returned result is an integer value in the range 0 to 59.

If the argument can be NULL, the result can be NULL; if the argument is NULL, the result is the NULLvalue.

Example

```
splice> VALUES( NOW, HOUR(NOW), MINUTE(NOW), SECOND(NOW) );
1          | 2          | 3          | 4
-----
2015-11-12 17:48:55.217    | 17         | 48         | 55.217
1 row selected
```

See Also

- » [About Data Types](#)
- » [TIMESTAMP data value](#)
- » [HOUR function](#)
- » [SECOND function](#)
- » [TIMESTAMP function](#)
- » [TIMESTAMPADD function](#)
- » [TIMESTAMPDIFF function](#)

MOD

MOD returns the remainder (modulus) of argument 1 divided by argument 2.

Syntax

```
mod(number, divisor)
```

number

The number for which you want to find the remainder after the division is performed.

divisor

The number by which you want to divide.

Results

The result is negative only if *number* is negative.

The result of the function is:

- » NULL if any argument is NULL.
- » SMALLINT.
- » SMALLINT.
- » SMALLINT.

The result can be NULL; if any argument is NULL, the result is the NULL value.

Examples

```

splice> VALUES MOD(37, 3);
1
-----
1

1 row selected

---select players with odd-numbered IDs:
splice> SELECT ID, Team, DisplayName
      FROM Players
     WHERE MOD(ID, 2) = 1
    ORDER BY ID;

```

ID	TEAM	DISPLAYNAME
1	Giants	Buddy Painter
3	Giants	John Purser
5	Giants	Mitch Duffer
7	Giants	Alex Paramour
9	Giants	Greg Brown
11	Giants	Kelly Tamlin
13	Giants	Andy Sussman
15	Giants	Elliot Andrews
17	Giants	Joseph Arkman
19	Giants	Jeremy Packman
21	Giants	Jason Pratter
23	Giants	Nathan Nickels
25	Giants	Reed Lister
27	Giants	Trevor Imhof
29	Giants	Charles Heillman
31	Giants	Thomas Hillman
33	Giants	Tam Lassiter
35	Giants	Mitch Lovell
37	Giants	Justin Oscar
39	Giants	Gary Kosovo
41	Giants	Steve Raster
43	Giants	Jason Lilliput
45	Giants	Cory Hammersmith
47	Giants	Barry Bochner
49	Cards	Yuri Milleton
51	Cards	Kelly Wacherman
53	Cards	Mitch Canepa
55	Cards	Pablo Bonjourno
57	Cards	Roger Green
59	Cards	Jeremy Johnson
61	Cards	Tad Philomen
63	Cards	Barry Morse
65	Cards	George Goomba
67	Cards	David Janssen
69	Cards	Edward Erdman
71	Cards	Don Allison
73	Cards	Carl Marin

```
75 |Cards      |Larry Lintos
77 |Cards      |Tim Lentleson
79 |Cards      |Carl Vanamos
81 |Cards      |Steve Mossely
83 |Cards      |Manny Stolanaro
85 |Cards      |Michael Hillson
87 |Cards      |Neil Gaston
89 |Cards      |Mo Grandosi
91 |Cards      |Mark Hasty
93 |Cards      |Stephen Tuvesco
```

```
47 rows selected
```

See Also

- » [About Data Types](#)
- » [DOUBLE PRECISION data type](#)

MONTH

The MONTH function returns the month part of a value.

Syntax

```
MONTH ( expression )
```

expression

An expression that can be a time, timestamp, or a valid character string representation of a time or timestamp.

Results

The returned result is an integer value in the range 1 to 12.

If the argument can be NULL, the result can be NULL; if the argument is NULL, the result is the NULL value.

Examples

Get the current date:

```
splice> VALUES(CURRENT_DATE);  
1  
-----  
2014-05-15
```

Now get the current month only:

```
splice> VALUES(MONTH(CURRENT_DATE));  
1  
-----  
5
```

Get the month of one week from now:

```
splice> VALUES(MONTH(CURRENT_DATE+7));  
1  
-----  
5
```

Select all players who were born in December:

```
splice> SELECT DisplayName, Team, BirthDate
      FROM Players
     WHERE MONTH(BirthDate)=12;
DISPLAYNAME          | TEAM        | BIRTHDATE
-----
Greg Brown           | Giants      | 1983-12-24
Reed Lister          | Giants      | 1986-12-16
Cameron Silliman    | Cards       | 1988-12-21
Edward Erdman        | Cards       | 1985-12-21
Taylor Trantula      | Cards       | 1987-12-17
Tam Croonster        | Cards       | 1980-12-19

6 rows selected
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)
- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [*Working with Dates* in the *Developer's Guide*](#)

MONTHNAME

The MONTHNAME function returns a character string containing month name from a date expression.

Syntax

```
MONTHNAME( dateExpr );
```

dateExpr

The date-time expression from which you wish to extract information.

Results

The returned month name is specific to the data source location; for English, the returned name will be in the range January through December, or Jan. through Dec. For a data source that uses German, the returned name will be in the range Januar through Dezember.

Examples

The following query displays the birth month of players:

```
splice> SELECT DisplayName, MONTHNAME(BirthDate) "Month"
   FROM Players
  WHERE ID<20
 ORDER BY MONTH(BirthDate);
DISPLAYNAME          | Month
-----
Bob Cranker        | January
Mitch Duffer       | January
Norman Aikman      | January
Jeremy Packman    | January
Buddy Painter      | March
Andy Sussman       | March
Billy Bopper        | April
Harry Pennello     | April
Alex Darba         | April
Kelly Tamlin       | June
Alex Paramour      | July
Mark Briste        | August
Elliot Andrews     | August
Joseph Arkman      | September
John Purser        | October
Craig McGawn       | October
Jason Minman       | November
Henry Socomy        | November
Greg Brown          | December

19 rows selected
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)

- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [Working with Dates](#) in the *Developer's Guide*

MONTH_BETWEEN

The MONTH_BETWEEN function returns the number of months between two dates.

Syntax

```
MONTH_BETWEEN( date1, date2 );
```

date1

The first date.

date2

The second date

Results

If date2 is later than date1, then the result is positive.

If date2 is earlier than date1, then the result is negative.

If date1 and date2 are either the same days of the month or both last days of months, then the result is always an integer.

Examples

```
splice> VALUES(MONTH_BETWEEN(CURRENT_DATE, DATE('2015-8-15')));
1
-----
3.0

splice> SELECT MIN(BirthDate) "Oldest",
    MAX(Birthdate) "Youngest",
    MONTH_BETWEEN(MIN(Birthdate), MAX(BirthDate)) "Months Between"
    FROM Players;
Oldest      | Youngest      | Months Between
-----
1975-07-14 | 1992-10-19 | 207.0
1 row selected
```

See Also

» [CURRENT_DATE function](#)

- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)
- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [Working with Dates](#) in the *Developer's Guide*

NEXT_DAY

The NEXT_DAY function returns the date of the next specified day of the week after a specified date.

Syntax

```
NEXT_DAY( source_date, day_of_week);
```

source_date

The source date.

day_of_week

The day of the week. This is the case-insensitive name of a day in the date language of your session. You can also specify day-name abbreviations, in which case any characters after the recognized abbreviation are ignored. For example, if you're using English, you can use the following values (again, the case of the characters is ignored):

Day Name	Abbreviation
Sunday	Sun
Monday	Mon
Tuesday	Tue
Wednesday	Wed
Thursday	Thu
Friday	Fri
Saturday	Sat

Results

This function returns the date of the first weekday, as specified by *day_of_week*, that is later than the specified date.

The return type is always DATE, regardless of the data type of the *source_date* parameter.

The return value has the same hours, minutes, and seconds components as does the *source_date* parameter value.

Examples

```
splice> values (NEXT_DAY(CURRENT_DATE, 'tuesday'));
1
-----
2014-09-23
1 row selected

splice> values (NEXT_DAY(CURRENT_DATE, 'monday'));
1
-----
2014-09-29
1 row selected

SELECT DisplayName, BirthDate, NEXT_DAY(BirthDate, 'sunday') as "FirstSunday"
  FROM Players
 WHERE ID < 20;
DISPLAYNAME          | BIRTHDATE | FirstSund&
-----
Buddy Painter        | 1987-03-27 | 1987-03-29
Billy Bopper         | 1988-04-20 | 1988-04-24
John Purser          | 1990-10-30 | 1990-11-04
Bob Cranker          | 1987-01-21 | 1987-01-25
Mitch Duffer         | 1991-01-15 | 1991-01-20
Norman Aikman        | 1982-01-05 | 1982-01-10
Alex Paramour        | 1981-07-02 | 1981-07-05
Harry Pennello       | 1983-04-13 | 1983-04-17
Greg Brown           | 1983-12-24 | 1983-12-25
Jason Minman         | 1983-11-06 | 1983-11-06
Kelly Tamlin          | 1990-06-16 | 1990-06-17
Mark Briste          | 1977-08-30 | 1977-09-04
Andy Sussman         | 1990-03-22 | 1990-03-25
Craig McGawn         | 1982-10-12 | 1982-10-17
Elliot Andrews        | 1989-08-21 | 1989-08-27
Alex Darba            | 1984-04-11 | 1984-04-15
Joseph Arkman        | 1984-09-21 | 1984-09-23
Henry Socomy          | 1989-11-17 | 1989-11-19
Jeremy Packman       | 1989-01-01 | 1989-01-01

19 rows selected
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)

- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NOW function](#)
- » [QUARTER function](#)
- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [Working with Dates in the *Developer's Guide*](#)

NOW

The NOW function returns the current date and time as a [TIMESTAMP](#) value.

Syntax

```
NOW();
```

Results

Returns the current date and time as a [TIMESTAMP](#) value.

Examples

```
splice> VALUES( NOW(), HOUR(NOW), MINUTE(NOW), SECOND(NOW) );
1          | 2          | 3          | 4
-----
2015-11-12 17:48:55.217    | 17        | 48        | 55.217
1 row selected
```

See Also

- » [CURRENT_DATE](#) function
- » [DATE](#) data type
- » [DATE](#) function
- » [DAY](#) function
- » [EXTRACT](#) function
- » [LASTDAY](#) function
- » [MONTH](#) function
- » [MONTH_BETWEEN](#) function
- » [MONTHNAME](#) function
- » [NEXTDAY](#) function
- » [QUARTER](#) function

- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [Working with Dates](#) in the *Developer's Guide*

NULLIF

The `NULLIF` function compares the values of two expressions; if they are equal, it returns `NULL`; otherwise, it returns the value of the first expression.

Syntax

```
NULLIF (expression1, expression2 )
```

expression1

The first .expression whose value you want to compare.

NOTE: You cannot specify the literal `NULL` for *expression1*.

expression2

The first .expression whose value you want to compare.

Results

The `NULLIF` function is logically similar to the following `CASE` expression:

```
CASE WHEN expression1 = expression2 THEN NULL ELSE expression1 END;
```

Example

```
splice> Select DisplayName "Position Player", NULLIF(Position, 'P') "Position"
      FROM Players
     WHERE MOD(ID, 2)=1
     ORDER BY Position;
Position Player          | Pos&
-----
Barry Morse              | 1B
David Janssen            | 1B
John Purser              | 2B
Kelly Tamlin             | 2B
Kelly Wacherman          | 2B
Mitch Duffer             | 3B
Mitch Canepa             | 3B
Buddy Painter            | C
Andy Sussman             | C
Yuri Milleton            | C
Edward Erdman            | C
Alex Paramour            | CF
Pablo Bonjourno          | CF
Jeremy Johnson            | CF
Tad Philomen              | CF
Nathan Nickels            | IF
George Goomba             | IF
Don Allison               | IF
Trevor Imhof              | LF
Elliot Andrews             | MI
Greg Brown                | OF
Jeremy Packman            | OF
Jason Pratter             | OF
Reed Lister               | OF
Roger Green               | OF
Charles Heillman           | NULL
Thomas Hillman             | NULL
Tam Lassiter              | NULL
Mitch Lovell               | NULL
Justin Oscar               | NULL
Gary Kosovo                | NULL
Steve Raster               | NULL
Jason Lilliput              | NULL
Cory Hammersmith           | NULL
Barry Bochner              | NULL
Carl Marin                 | NULL
Larry Lintos               | NULL
Tim Lentleson              | NULL
Carl Vanamos               | NULL
Steve Mossely               | NULL
Manny Stolanaro             | NULL
Michael Hillson              | NULL
Neil Gaston                 | NULL
Mo Grandosi                | NULL
Mark Hasty                  | NULL
```

Stephen Tuvesco	NULL
Joseph Arkman	UT

47 rows selected

See Also

» [CASE expression](#)

NVL

The NVL function returns the first non-NULL expression from a list of expressions.

You can also use NVL as a variety of a CASE expression. For example:

```
NVL( expression_1, expression_2,...expression_n);
```

is equivalent to:

```
CASE WHEN expression_1 IS NOT NULL THEN expression_1
      ELSE WHEN expression_1 IS NOT NULL THEN expression_2
      ...
      ELSE expression_n;
```

Syntax

```
NVL ( expression1, expression2 [, expressionN]* )
```

expression1

An expression.

expression1

An expression.

expressionN

You can specify more than two arguments; **you MUST specify at least two arguments.**

Usage

VALUE is a synonym for NVL that is accepted by Splice Machine, but is not recognized by the SQL standard.

Results

The result is NULL only if all of the arguments are NULL.

An error occurs if all of the parameters of the function call are dynamic.

Example

```
-- create table with three different integer types
splice> SELECT ID, FldGames, PassedBalls, WildPitches, Pickoffs,
  NVL(PassedBalls, WildPitches, Pickoffs) as "FirstNonNull"
  FROM Fielding
 WHERE FldGames>50
 ORDER BY ID;
ID      |FLDGA&|PASSE&|WILDP&|PICKO&|First&
-----
1       | 142   | 4      | 20     | 0      | 4
2       | 131   | NULL   | NULL   | NULL   | NULL
3       | 99    | NULL   | NULL   | NULL   | NULL
4       | 140   | NULL   | NULL   | NULL   | NULL
5       | 142   | NULL   | NULL   | NULL   | NULL
6       | 88    | NULL   | NULL   | NULL   | NULL
7       | 124   | NULL   | NULL   | NULL   | NULL
8       | 51    | NULL   | NULL   | NULL   | NULL
9       | 93    | NULL   | NULL   | NULL   | NULL
10      | 79    | NULL   | NULL   | NULL   | NULL
39      | 73    | NULL   | NULL   | 0      | 0
40      | 52    | NULL   | NULL   | 0      | 0
41      | 70    | NULL   | NULL   | 2      | 2
42      | 55    | NULL   | NULL   | 0      | 0
43      | 77    | NULL   | NULL   | 0      | 0
46      | 67    | NULL   | NULL   | 0      | 0
49      | 134   | 4      | 34     | 2      | 4
50      | 119   | NULL   | NULL   | NULL   | NULL
51      | 147   | NULL   | NULL   | NULL   | NULL
52      | 148   | NULL   | NULL   | NULL   | NULL
53      | 152   | NULL   | NULL   | NULL   | NULL
54      | 64    | NULL   | NULL   | NULL   | NULL
55      | 93    | NULL   | NULL   | NULL   | NULL
56      | 147   | NULL   | NULL   | NULL   | NULL
57      | 85    | NULL   | NULL   | NULL   | NULL
58      | 62    | NULL   | NULL   | NULL   | NULL
59      | 64    | NULL   | NULL   | NULL   | NULL
62      | 53    | 1      | 11     | 0      | 1
64      | 59    | NULL   | NULL   | NULL   | NULL
81      | 76    | NULL   | NULL   | 0      | 0
82      | 71    | NULL   | NULL   | 1      | 1
84      | 68    | NULL   | NULL   | 0      | 0
92      | 81    | NULL   | NULL   | 3      | 3

33 rows selected
```

PI

The PI function returns a value that is closer than any other value to pi. The constant pi is the ratio of the circumference of a circle to the diameter of a circle.

Syntax

```
PI( )
```

Syntax

The data type of the returned value is a [DOUBLE PRECISION](#) number.

Example

```
splice> VALUES PI();  
1  
-----  
3.14159265358793  
1 row selected
```

See Also

» [DOUBLE PRECISION](#) data type

QUARTER

The QUARTER function returns an integer value representing the quarter of the year from a date expression.

Syntax

```
QUARTER( dateExpr );
```

dateExpr

The date-time expression from which you wish to extract information.

Results

The returned week number is in the range 1 to 4. January 1 through March 31 is Quarter 1.

Examples

```
splice> VALUES QUARTER('2009-01-02 11:22:33.04');
1
-----
1
1 row selected

splice> SELECT DisplayName, BirthDate, Quarter(BirthDate) "Quarter"
  FROM Players
 WHERE ID<20
 ORDER BY "Quarter", BirthDate;
DISPLAYNAME          |BIRTHDATE |Quarter
-----
Norman Aikman        |1982-01-05|1
Bob Cranker          |1987-01-21|1
Buddy Painter         |1987-03-27|1
Jeremy Packman       |1989-01-01|1
Andy Sussman         |1990-03-22|1
Mitch Duffer          |1991-01-15|1
Harry Pennello        |1983-04-13|2
Alex Darba            |1984-04-11|2
Billy Bopper          |1988-04-20|2
Kelly Tamlin          |1990-06-16|2
Mark Briste           |1977-08-30|3
Alex Paramour         |1981-07-02|3
Joseph Arkman         |1984-09-21|3
Elliot Andrews        |1989-08-21|3
Craig McGawn          |1982-10-12|4
Jason Minman          |1983-11-06|4
Greg Brown             |1983-12-24|4
Henry Socomy           |1989-11-17|4
John Purser           |1990-10-30|4

19 rows selected
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)

- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [*Working with Dates*](#) in the *Developer's Guide*

RADIANS

The RADIANS function converts a specified number from degrees to radians.

The specified number is an angle measured in degrees, which is converted to an approximately equivalent angle measured in radians. The specified number must be a [DOUBLE PRECISION](#) number.

NOTE: The conversion from degrees to radians is not exact.

The data type of the returned value is a DOUBLE PRECISION number.

Syntax

```
RADIANS ( number )
```

Example

```
splice> VALUES RADIANS(90);
1
-----
1.5707963267948966

1 row selected
```

See Also

- » [DOUBLE PRECISION](#) data type
- » [ACOS](#) function
- » [ASIN](#) function
- » [ATAN](#) function
- » [ATAN2](#) function
- » [COS](#) function
- » [COSH](#) function
- » [COT](#) function
- » [DEGREES](#) function
- » [SIN](#) function

» [SINH function](#)

» [TAN function](#)

» [TANH function](#)

RAND

The RAND function returns a random number given a seed number

The RAND function returns an [INTEGER](#) seed number.

Syntax

```
RAND( seed )
```

Example

```
splice> VALUES RAND(13);  
1  
-----  
0.7298032243379924  
1 row selected
```

See Also

- » [About Data Types](#)
- » [DOUBLE PRECISION](#) data type

RANDOM

The RANDOM function returns a random number.

The RANDOM function returns an [INTEGER](#) seed number.

Syntax

```
RANDOM()
```

Example

```
splice> VALUES RANDOM();
1
-----
0.2826393098638572

1 row selected
```

See Also

- » [About Data Types](#)
- » [DOUBLE PRECISION](#) data type

RANK()

`RANK()` is a *ranking function* that returns the rank of a value within the ordered partition of values defined by its `OVER` clause. Ranking functions are a subset of window functions.

Syntax

```
RANK() OVER ( overClause )
```

`overClause`

See the [OVER clause documentation](#).

NOTE: Ranking functions such as `RANK` must include an `ORDER BY` clause in the `OVER` clause. This is because the ranking is calculated based on the ordering.

Results

The resulting data type is [BIGINT](#).

Usage

The `RANK()` and `DENSE_RANK()` analytic functions are very similar. The difference shows up when there are multiple input rows that have the same ranking value. When that happens:

- » The `RANK()` function can generate non-consecutive ranking result values: if values in the ranking column are the same, they receive the same rank; however, the next number in the ranking sequence is then skipped, which means that `RANK` can return non-consecutive numbers.
- » The `DENSE_RANK()` function always returns consecutive rankings: if values in the ranking column are the same, they receive the same rank, and the next number in the ranking sequence is then used to rank the row or rows that follow.

Here's a simple example that shows the ranking produced by the two functions for input with duplicate values to illustrate that difference:

Value	RANK	DENSE_RANK
a	1	1
a	1	1
a	1	1
b	4	2
c	5	3
c	5	3
d	7	4
e	8	5

Example

The following query ranks the salaries of players, per team, whose salary is at least \$1 million.

```

SELECT DisplayName, Team, Season, Salary,
       RANK() OVER (PARTITION BY Team ORDER BY Salary Desc) "RANK"
  FROM Players JOIN Salaries ON Salaries.ID=Players.ID
 WHERE Salary>999999 AND Season=2015;

```

DISPLAYNAME	TEAM	SEASON	SALARY	RANK
Mitch Hassleman	Cards	2015	17000000	1
Yuri Miletton	Cards	2015	15200000	2
James Grasser	Cards	2015	9375000	3
Jack Hellman	Cards	2015	8300000	4
Larry Lintos	Cards	2015	7000000	5
Jeremy Johnson	Cards	2015	4125000	6
Mitch Canepa	Cards	2015	3750000	7
Mitch Brandon	Cards	2015	3500000	8
Robert Cohen	Cards	2015	3000000	9
James Woegren	Cards	2015	2675000	10
Sam Culligan	Cards	2015	2652732	11
Barry Morse	Cards	2015	2379781	12
Michael Rastono	Cards	2015	2000000	13
Carl Vanamos	Cards	2015	2000000	13
Alex Wister	Cards	2015	1950000	15
Pablo Bonjourno	Cards	2015	1650000	16
Jonathan Pearlman	Cards	2015	1500000	17
Jan Bromley	Cards	2015	1200000	18
Martin Cassman	Giants	2015	20833333	1
Harry Pennello	Giants	2015	18500000	2
Tam Lassiter	Giants	2015	18000000	3
Buddy Painter	Giants	2015	17277777	4
Thomas Hillman	Giants	2015	12000000	5
Alex Paramour	Giants	2015	10250000	6
Jack Peepers	Giants	2015	9000000	7
Mark Briste	Giants	2015	8000000	8
Marcus Bamburger	Giants	2015	6950000	9
Jalen Ardson	Giants	2015	6000000	10
Steve Raster	Giants	2015	6000000	10
Sam Castleman	Giants	2015	5000000	12
Craig McGawn	Giants	2015	4800000	13
Norman Aikman	Giants	2015	4000000	14
Randy Varner	Giants	2015	4000000	14
Jason Lilliput	Giants	2015	4000000	14
Billy Bopper	Giants	2015	3600000	17
Greg Brown	Giants	2015	3600000	17
Mitch Lovell	Giants	2015	3578825	19
Bob Cranker	Giants	2015	3175000	20
Yuri Piamam	Giants	2015	2100000	21
Joseph Arkman	Giants	2015	1450000	22
Trevor Imhof	Giants	2015	1100000	23
Jason Minman	Giants	2015	1000000	24

42 rows selected

Here's the same query using `DENSE_RANK` instead of `RANK`. Note how tied rankings are handled differently:

```

SELECT DisplayName, Team, Season, Salary,
       DENSE_RANK() OVER (PARTITION BY Team ORDER BY Salary Desc) "RANK"
  FROM Players JOIN Salaries ON Salaries.ID=Players.ID
 WHERE Salary>999999 AND Season=2015;

```

DISPLAYNAME	TEAM	SEASON	SALARY	RANK
Mitch Hassleman	Cards	2015	17000000	1
Yuri Miletton	Cards	2015	15200000	2
James Grasser	Cards	2015	9375000	3
Jack Hellman	Cards	2015	8300000	4
Larry Lintos	Cards	2015	7000000	5
Jeremy Johnson	Cards	2015	4125000	6
Mitch Canepa	Cards	2015	3750000	7
Mitch Brandon	Cards	2015	3500000	8
Robert Cohen	Cards	2015	3000000	9
James Woegren	Cards	2015	2675000	10
Sam Culligan	Cards	2015	2652732	11
Barry Morse	Cards	2015	2379781	12
Michael Rastono	Cards	2015	2000000	13
Carl Vanamos	Cards	2015	2000000	13
Alex Wister	Cards	2015	1950000	14
Pablo Bonjourno	Cards	2015	1650000	15
Jonathan Pearlman	Cards	2015	1500000	16
Jan Bromley	Cards	2015	1200000	17
Martin Cassman	Giants	2015	20833333	1
Harry Pennello	Giants	2015	18500000	2
Tam Lassiter	Giants	2015	18000000	3
Buddy Painter	Giants	2015	17277777	4
Thomas Hillman	Giants	2015	12000000	5
Alex Paramour	Giants	2015	10250000	6
Jack Peepers	Giants	2015	9000000	7
Mark Briste	Giants	2015	8000000	8
Marcus Bamburger	Giants	2015	6950000	9
Jalen Ardson	Giants	2015	6000000	10
Steve Raster	Giants	2015	6000000	10
Sam Castleman	Giants	2015	5000000	11
Craig McGawn	Giants	2015	4800000	12
Norman Aikman	Giants	2015	4000000	13
Randy Varner	Giants	2015	4000000	13
Jason Lilliput	Giants	2015	4000000	13
Billy Bopper	Giants	2015	3600000	14
Greg Brown	Giants	2015	3600000	14
Mitch Lovell	Giants	2015	3578825	15
Bob Cranker	Giants	2015	3175000	16
Yuri Piamam	Giants	2015	2100000	17
Joseph Arkman	Giants	2015	1450000	18
Trevor Imhof	Giants	2015	1100000	19
Jason Minman	Giants	2015	1000000	20

42 rows selected

See Also

- » Window and Aggregate functions
- » **BIGINT** data type
- » **DENSE_RANK** function
- » **OVER** clause
- » [Working with Dates](#) in the *Developer's Guide*

REGEXP_LIKE Operator

The REGEXP_LIKE operator returns `true` if the string matches the regular expression. This function is similar to the `LIKE` predicate, except that it uses regular expressions rather than simple wildcard character matching.

Syntax

```
REGEXP_LIKE( sourceString, patternString )
```

sourceString

The character expression to match against the regular expression.

patternString

The regular expression string used to search for a match in *sourceString*.

The pattern is a `java.util.regex` pattern. You can find documentation for the JDK 8 version here: <http://docs.oracle.com/javase/8/docs/api/java/util/regex/package-summary.html>.

Results

Returns true if the *sourceString* you are testing matches the specified regular expression in *patternString*.

Examples

The following query finds all players whose name begins with `Ste`:

```
splice> SELECT DisplayName
      FROM Players
     WHERE REGEXP_LIKE(DisplayName, '^Ste.*');

DISPLAYNAME
-----
Steve Raster
Steve Mossely
Stephen Tuvesco

3 rows selected
```

See Also

» About Data Types

- » [CONCATENATION operator](#)
- » [INITCAP function](#)
- » [INSTR function](#)
- » [LCASE function](#)
- » [LENGTH function](#)
- » [LTRIM function](#)
- » [REPLACE function](#)
- » [RTRIM function](#)
- » [SUBSTR function](#)
- » [TRIM function](#)
- » [UCASE function](#)

REPLACE

The REPLACE function replaces all occurrences of a substring within a string and returns the new string.

Syntax

```
REPLACE(subjectStr, searchStr, replaceStr)
```

subjectStr

The string you want modified. This can be a literal string or a reference to a `char` or `varchar` value.

searchStr

The substring to replace within *subjectStr*. This can be a literal string or a reference to a `char` or `varchar` value.

replaceStr

The replacement substring. This can be a literal string or a reference to a `char` or `varchar` value.

Results

A string value.

Examples

The first examples shows the players on each team with averages greater than .300. The second example shows the result of replacing the team of those players with averages greater than 0.300 who play on one team (the Cards):

```

splice> SELECT DisplayName, Average, Team
      FROM Players JOIN Batting on Players.ID=Batting.ID
      WHERE Average > 0.300 AND Games>50;

DISPLAYNAME          | AVERAGE | TEAM
-----
Buddy Painter        | 0.31777 | Giants
John Purser          | 0.31151 | Giants
Kelly Tamlin         | 0.30337 | Giants
Stan Post            | 0.30472 | Cards

4 rows selected

splice> SELECT DisplayName, Average,
      REPLACE(Team, 'Cards', 'Giants') "TRADED"
      FROM PLAYERS JOIN Batting ON Players.ID=Batting.ID
      WHERE Team='Cards' AND Average > 0.300 AND Games > 50;

DISPLAYNAME          | AVERAGE | TRADED
-----
Stan Post            | 0.30472 | Giants

1 row selected

```

See Also

- » [About Data Types](#)
- » [Concatenation operator](#)
- » [INITCAP function](#)
- » [INSTR function](#)
- » [LCASE function](#)
- » [LENGTH function](#)
- » [LOCATE function](#)
- » [LTRIM function](#)
- » [REGEX_LIKE operator](#)
- » [RTRIM function](#)
- » [SUBSTR function](#)
- » [TRIM function](#)
- » [UCASE function](#)

ROWID

ROWID is a *pseudocolumn* that uniquely defines a single row in a database table.

The term pseudocolumn is used because you can refer to ROWID in the [WHERE](#) clauses of a query as you would refer to a column stored in your database; the difference is you cannot insert, update, or delete ROWID values.

The ROWID value for a given row in a table remains the same for the life of the row, with one exception: the ROWID may change if the table is an index organized table and you change its primary key.

Syntax

ROWID

Usage

You can use a ROWID value to refer to a row in a table in the [WHERE](#) clauses of a query. These values have several valuable uses:

- » They are the fastest way to access a single row.
- » They are a built-in, unique identifier for every row in a table.
- » They provide information about how the rows in a table are stored.

Some important notes about ROWID values:

- » Do not use ROWID as the primary key of a table.
- » The ROWID of a deleted row can later be reassigned to a new row.
- » A ROWID value is associated with a table row when the row is created.
- » ROWID values are unique within a table, but not necessarily unique within a database.
- » If you delete and re-import a row in a table, the ROWID may change.
- » The ROWID value for a row may change if the row is in an index organized table and you change the table's primary key.

Using ROWID with JDBC

You can access ROWID with JDBC result sets; for example:

```
( ) ResultSet.getRowId(int);
```

You can also use ROWID in JDBC queries; for example:

```
() CallableStatement.setRowId(int, RowId);
() PreparedStatement.setRowId(int, RowId);
```

Examples

This statement selects the unique row address and salary of all records in the employees database in the engineering department:

```
splice> SELECT ROWID, DisplayName, Position
      FROM Players
     WHERE Team='Giants' and Position='OF';

ROWID          | DISPLAYNAME           | POS&
-----|-----
89            | Greg Brown            | OF
93            | Jeremy Packman        | OF
95            | Jason Pratter          | OF
99            | Reed Lister            | OF

4 rows selected
```

This statement updates column c in all rows in which column b equals 10:

```
UPDATE mytable SET c=100 WHERE rowid=(SELECT rowid FROM mytable WHERE b=10);
```

See Also

- » [SELECT expression](#)
- » [SELECT statement](#)
- » [UPDATE statement](#)
- » [WHERE clause](#)

ROW_NUMBER

ROW_NUMBER() is a *ranking function* that numbers the rows within the ordered partition of values defined by its OVER clause. Ranking functions are a subset of window functions.

Syntax

```
ROW_NUMBER() OVER ( overClause )
```

overClause

See the [OVER](#) clause documentation.

NOTE: Ranking functions such as ROW_NUMBER must include an [ORDER BY](#) clause in the OVER clause. This is because the ranking is calculated based on the ordering.

Results

The resulting data type is [BIGINT](#).

Example

The following query ranks the salaries of players on the Cards whose salaries are at least \$1 million:

```
splice> SELECT DisplayName, Salary,
    ROW_NUMBER() OVER (PARTITION BY Team ORDER BY Salary DESC) "RowNum"
    FROM Players JOIN Salaries ON Players.ID=Salaries.ID
    WHERE Team='Cards' and Salary>999999;
```

DISPLAYNAME	SALARY	RowNum
Mitch Hassleman	17000000	1
Yuri Milleton	15200000	2
James Grasser	9375000	3
Jack Hellman	8300000	4
Larry Lintos	7000000	5
Jeremy Johnson	4125000	6
Mitch Canepa	3750000	7
Mitch Brandon	3500000	8
Robert Cohen	3000000	9
James Woegren	2675000	10
Sam Culligan	2652732	11
Barry Morse	2379781	12
Michael Rastono	2000000	13
Carl Vanamos	2000000	14
Alex Wister	1950000	15
Pablo Bonjourno	1650000	16
Jonathan Pearlman	1500000	17
Jan Bromley	1200000	18

18 rows selected

See Also

- » [Window and Aggregate functions](#)
- » [BIGINT data type](#)
- » [OVER clause](#)
- » [OVER clause](#)
- » [Using Window Functions](#) in the *Developer Guide*.

RTRIM

RTRIM removes blanks from the end of a character string expression.

Syntax

```
RTRIM(CharacterExpression)
```

CharacterExpression

A `LONG VARCHAR` data type, any built-in type that is implicitly converted to a string.

Results

A character string expression. If the *CharacterExpression* evaluates to `NULL`, this function returns `NULL`.

Examples

```
splice> VALUES RTRIM('      Space Case      ');
1
-----
Space Case      --- This is the string '      Space Case'
```

See Also

- » [About Data Types](#)
- » [Concatenation operator](#)
- » [INITCAP function](#)
- » [INSTR function](#)
- » [LCASE function](#)
- » [LENGTH function](#)
- » [LOCATE function](#)
- » [LTRIM function](#)
- » [REGEX_LIKE operator](#)
- » [REPLACE function](#)
- » [SUBSTR function](#)

» [TRIM function](#)

» [UCASE function](#)

SECOND

The SECOND function returns the seconds part of a value.

Syntax

```
SECOND( expression )
```

expression

An expression that can be a time, timestamp, or a valid character string representation of a time or timestamp.

Results

The returned result is an integer value in the range 0 to 59.

If the argument can be NULL, the result can be NULL; if the argument is NULL, the result is the NULLvalue.

Example

```
splice> VALUES( NOW(), HOUR(NOW), MINUTE(NOW), SECOND(NOW) );
1          | 2          | 3          | 4
-----
2015-11-12 17:48:55.217    | 17         | 48         | 55.217
1 row selected
```

See Also

- » [About Data Types](#)
- » [TIMESTAMP data value](#)
- » [HOUR function](#)
- » [MINUTE function](#)
- » [TIMESTAMP function](#)
- » [TIMESTAMPADD function](#)
- » [TIMESTAMPDIFF function](#)

SESSION_USER

When used outside stored routines, `CURRENT_USER`, `USER`, and `SESSION_USER` all return the authorization identifier of the user who created the SQL session.

`SESSION_USER` also always returns this value when used within stored routines.

If used within a stored routine created with `EXTERNAL SECURITY DEFINER`, however, `CURRENT_USER` and `USER` return the authorization identifier of the user that owns the schema of the routine. This is usually the creating user, although the database owner could be the creator as well.

For information about definer's and invoker's rights, see [CREATE FUNCTION](#) statement.

Syntax

```
SESSION_USER
```

Example

```
VALUES SESSION_USER;
1
-----
SPLICE
1 row selected
```

See Also

- » [CURRENT_USER function](#)
- » [USER function](#)
- » [CREATE_FUNCTION statement](#)
- » [CREATE_PROCEDURE statement](#)

SIGN

The SIGN function returns the sign of the specified number.

Syntax

```
SIGN ( number )
```

number

A [DOUBLE PRECISION](#) number that specifies the value whose sign you want.

Results

The data type of the returned value is [INTEGER](#):

- » If the specified number is NULL, the result of this function is NULL.
- » If the specified number is zero (0), the result of this function is zero (0).
- » If the specified number is greater than zero (0), the result of this function is plus one (+1).
- » If the specified number is less than zero (0), the result of this function is minus one (-1).

Example

```
splice> VALUES( SIGN(84.4), SIGN(-85.5), SIGN(0), SIGN(NULL) );
1          | 2          | 3          | 4
-----
1          | -1         | 0          | NULL
1 row selected
```

See Also

- » [DOUBLE PRECISION](#) data type

SIN

The SIN function returns the sine of a specified number.

Syntax

```
SIN ( number )
```

number

A DOUBLE PRECISION number that specifies the angle, in radians, for which you want the sine computed.

Results

The data type of the returned value is a DOUBLE PRECISION number.

If *number* is NULL, the result of the function is NULL.

If *number* is 0, the result of the function is 0.

Example

```
splice> VALUES SIN(84.4);
1
-----
0.4104993826174394

1 row selected
```

See Also

- » [DOUBLE PRECISION data type](#)
- » [ACOS function](#)
- » [ASIN function](#)
- » [ATAN function](#)
- » [ATAN2 function](#)
- » [COS function](#)
- » [COSH function](#)

- » [COT function](#)
- » [DEGREES function](#)
- » [RADIANS function](#)
- » [SINH function](#)
- » [TAN function](#)
- » [TANH function](#)

SINH

The **SINH** function returns the hyperbolic sine of a specified number.

Syntax

```
SINH ( number )
```

number

A **DOUBLE PRECISION** number that specifies the angle, in radians, for which you want the hyperbolic sine computed.

Results

The data type of the returned value is a **DOUBLE PRECISION** number.

If *number* is **NULL**, the result of the function is **NULL**.

If *number* is 0, the result of the function is 0.

Example

```
splice> VALUES SINH(84.4);
1
-----
2.2564425307671042E36

1 row selected
```

See Also

- » [DOUBLE PRECISION data type](#)
- » [ACOS function](#)
- » [ASIN function](#)
- » [ATAN function](#)
- » [ATAN2 function](#)
- » [COS function](#)

- » [COSH function](#)
- » [COT function](#)
- » [DEGREES function](#)
- » [RADIANS function](#)
- » [SIN function](#)
- » [TAN function](#)
- » [TANH function](#)

SMALLINT

The **SMALLINT** function returns a small integer representation of a number or character string, in the form of a small integer constant.

Syntax

```
SMALLINT ( NumericExpression | CharacterExpression )
```

NumericExpression

An expression that returns a value of any built-in numeric data type.

CharacterExpression

An expression that returns a character string value of length not greater than the maximum length of a character constant. Leading and trailing blanks are eliminated and the resulting string must conform to the rules for forming an SQL integer constant. The value of the constant must be in the range of small integers. The character string cannot be a long string.

Results

The result of the function is a **SMALLINT**. If the argument can be **NULL**, the result can be **NULL**. If the argument is **NULL**, the result is the **NULL**value.

If the argument is a *NumericExpression*, the result is the same number that would occur if the argument were assigned to a small integer column or variable. If the whole part of the argument is not within the range of small integers, an error occurs. The decimal part of the argument is truncated if present.

If the argument is a *CharacterExpression*, the result is the same number that would occur if the corresponding integer constant were assigned to a small integer column or variable.

Examples

Using the `Pitching` table from our Doc Examples database, select the `Era` column in big integer form for further processing in the application:

```
splice> SELECT ID, SMALLINT(Era) "ERA"
      FROM Pitching
     WHERE MOD(ID,2) = 0;
ID      | ERA
-----
28      | 2
30      | 4
32      | 3
34      | 5
36      | 3
38      | 5
40      | 5
42      | 2
44      | 5
46      | 2
48      | 5
72      | 2
74      | 3
76      | 2
78      | 3
80      | 1
82      | 3
84      | 2
86      | 2
88      | 0
90      | 2
92      | 2
94      | 2

23 rows selected
```

See Also

- » [About Data Types](#)

SQRT

The SQRT function returns the square root of a floating point number.

NOTE: To execute SQRT on data types other than floating point numbers, you must first cast them to floating point types.

Syntax

```
SQRT(FloatingPointExpression)
```

FloatingPointExpression

A `DOUBLE PRECISION` number.

Results

The return type for SQRT is the type of the input parameter value.

Examples

```

splice> VALUES sqrt(3421E+09);
1
-----
1849594.5501649815

1 row selected

-- Shows using SQRT on a SMALLINT column
splice> select Strikeouts, SQRT(Strikeouts) "SQRT"
   FROM Batting
  WHERE Strikeouts > 50
  ORDER BY Strikeouts;
STRIK& | SQRT
-----
52    | 7.211102550927978
56    | 7.483314773547883
59    | 7.681145747868608
59    | 7.681145747868608
59    | 7.681145747868608
76    | 8.717797887081348
90    | 9.486832980505138
93    | 9.643650760992955
95    | 9.746794344808963
96    | 9.797958971132712
110   | 10.488088481701515
111   | 10.535653752852738
119   | 10.908712114635714
121   | 11.0
147   | 12.12435565298214
151   | 12.288205727444508

16 rows selected

splice> SELECT ID, FieldingIndependent, SQRT(FieldingIndependent) "SQRT"
   FROM Pitching
  WHERE Mod(ID, 2)=1;
ID     | FIELDI& | SQRT
-----
29    | 4.02    | 2.004993765576342
31    | 4.53    | 2.1283796653792764
33    | 4.29    | 2.071231517720798
35    | 4.83    | 2.1977260975835913
37    | 3.90    | 1.9748417658131499
39    | 4.02    | 2.004993765576342
41    | 1.91    | 1.3820274961085253
43    | 3.36    | 1.833030277982336
45    | 4.81    | 2.1931712199461306
47    | 3.13    | 1.7691806012954132
73    | 3.21    | 1.7916472867168918
75    | 3.44    | 1.8547236990991407
77    | 4.53    | 2.1283796653792764

```

79	3.74	1.9339079605813716
81	3.78	1.944222209522358
83	2.76	1.6613247725836149
85	5.39	2.32163735324878
89	8.38	2.894822965226026
91	4.63	2.151743479135001
93	3.82	1.9544820285692064

20 rows selected

See Also

- » [About Data Types](#)

STDDEV_POP

The STDDEV_POP() function returns the population standard deviation of a set of numeric values.

It returns NULL if no matching row is found.

Syntax

```
STDDEV_POP ( [ DISTINCT | ALL ] expression )
```

DISTINCT

If this qualifier is specified, duplicates are eliminated

ALL

If this qualifier is specified, all duplicates are retained. This is the default value.

expression

An expression that evaluates to a numeric data type: [SMALLINT](#).

The expression can contain multiple column references or expressions, but it cannot contain another aggregate or subquery, and it must evaluate to an ANSI SQL numeric data type. This means that you can call methods that evaluate to ANSI SQL data types.

If an expression evaluates to NULL, the aggregate skips that value.

Results

This function returns a double-precision number.

If the input expression consists of a single value, the result of the function is NULL, not 0.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

Example

The following example shows computing the average, population standard deviation, and sample standard deviation from our Salaries table:

```
splice> SELECT AVG(Salary) as AvgSalary, STDDEV_POP(Salary) AS PopStdDev, STDDEV_SAMP(Salary) As SampStdDev FROM Salaries;
AVGSALARY          | POPSTDDEV           | SAMPSTDDEV
-----
2949737            | 4694155.715951055 | 4719325.63212163
1 row selected
```

STDDEV_SAMP

The STDDEV_POP() function returns the sample standard deviation of a set of numeric values.

It returns NULL if no matching row is found.

Syntax

```
STDDEV_SAMP ( [ DISTINCT | ALL ] expression )
```

DISTINCT

If this qualifier is specified, duplicates are eliminated

ALL

If this qualifier is specified, all duplicates are retained. This is the default value.

expression

An expression that evaluates to a numeric data type: [SMALLINT](#).

The expression can contain multiple column references or expressions, but it cannot contain another aggregate or subquery, and it must evaluate to an ANSI SQL numeric data type. This means that you can call methods that evaluate to ANSI SQL data types.

If an expression evaluates to NULL, the aggregate skips that value.

Results

This function returns a double-precision number.

If the input expression consists of a single value, the result of the function is NULL, not 0.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

Example

The following example shows computing the average, population standard deviation, and sample standard deviation from our Salaries table:

```
splice> SELECT AVG(Salary) as AvgSalary, STDDEV_POP(Salary) AS PopStdDev, STDDEV_SAMP(Salary) As SampStdDev FROM Salaries;
AVGSALARY          | POPSTDDEV           | SAMPSTDDEV
-----
2949737            | 4694155.715951055 | 4719325.63212163
1 row selected
```

SUBSTR

The SUBSTR function extracts and returns a portion of a character string or bit string expression, starting at the specified character or bit position. You can specify the number of characters or bits you want returned, or use the default length, which is to extract from the specified starting position to the end of the string.

Syntax

```
SUBSTR( { CharacterExpression } ,
          StartPosition [ , LengthOfSubstring ] )
```

CharacterExpression

A CHAR, VARCHAR, or LONG VARCHAR data type or any built-in type that is implicitly converted to a string (except a bit expression).

StartPosition

An integer expression; for character expressions, this is the starting character position of the returned substring. For bit expressions, this is the bit position of the returned substring.

The first character or bit has a *StartPosition* of 1. If you specify 0, Splice Machine assumes that you mean 1.

If the *StartPosition* is positive, it refers to the position from the start of the source expression (counting the first character as 1) to the beginning of the substring you want extracted. The *StartPosition* value cannot be a negative number.

LengthOfSubstring

An optional integer expression that specifies the length of the extracted substring; for character expressions, this is the number of characters to return. For bit expressions, this is the number of bits to return.

If this value is not specified, then SUBSTR extracts a substring of the expression from the *StartPosition* to the end of the source expression.

If *LengthOfString* is specified, SUBSTR returns a VARCHAR or VARBIT of length *LengthOfString* starting at the *StartPosition*.

The SUBSTR function returns an error if you specify a negative number for the parameter *LengthOfString*.

Results

For character string expressions, the result type is a [VARCHAR](#) value.

The length of the result is the maximum length of the source type.

Examples

The following query extracts the first four characters of each player's name, and then extracts the remaining characters:

```

splice> SELECT DisplayName,
    SUBSTR(DisplayName, 1, 4) "1to4",
    SUBSTR(DisplayName, 4) "5ToEnd"
    FROM Players
    WHERE ID < 11;
DISPLAYNAME          | 1To4 | 5ToEnd
-----
Buddy Painter        | Budd|dy Painter
Billy Bopper         | Bill|ly Bopper
John Purser          | John|n Purser
Bob Cranker          | Bob | Cranker
Mitch Duffer         | Mitc|ch Duffer
Norman Aikman        | Norm|man Aikman
Alex Paramour        | Alex|x Paramour
Harry Pennello        | Harr|ry Pennello
Greg Brown            | Greg|g Brown
Jason Minman          | Jaso|on Minman

10 rows selected

```

See Also

- » [About Data Types](#)
- » [Concatenation operator](#)
- » [INITCAP function](#)
- » [INSTR function](#)
- » [LCASE function](#)
- » [LENGTH function](#)
- » [LOCATE function](#)
- » [LTRIM function](#)
- » [REGEX_LIKE operator](#)
- » [REPLACE function](#)
- » [RTRIM function](#)
- » [TRIM function](#)
- » [UCASE function](#)

SUM

SUM returns the sum of values of an expression over a set of rows. You can use it as an window (analytic) function.

The SUM function function takes as an argument any numeric data type or any non-numeric data type that can be implicitly converted to a numeric data type. The function returns the same data type as the numeric data type of the argument.

Syntax

```
SUM ( [ DISTINCT | ALL ] Expression )
```

DISTINCT

If this qualifier is specified, duplicates are eliminated

If you specify DISTINCT in the analytic version of SUM, the OVER clause for your window function cannot include an ORDER BY clause or a *frame clause*.

ALL

If this qualifier is specified, all duplicates are retained. This is the default value.

Expression

An expression that evaluates to a numeric data type: [SMALLINT](#).

An *Expression* can contain multiple column references or expressions, but it cannot contain another aggregate or subquery.

If an *Expression* evaluates to NULL, the aggregate skips that value.

Usage

The *Expression* can contain multiple column references or expressions, but it cannot contain another aggregate or subquery. It must evaluate to a built-in numeric data type. If an expression evaluates to NULL, the aggregate skips that value.

Only one DISTINCT aggregate expression per *Expression* is allowed. For example, the following query is not valid:

```
-- query not allowed
SELECT AVG (DISTINCT flying_time),
       SUM (DISTINCT miles)
  FROM Flights;
```

Note that specifying DISTINCT can result in a different value, since a smaller number of values may be summed. For example, if a column contains the values 1, 1, 1, 1, and 2, SUM(col) returns a greater value than SUM(DISTINCT col).

Results

The resulting data type is the same as the expression on which it operates (it might overflow).

Aggregate Examples

These queries compute the total of all salaries for all teams, and then the total for each individually.

```
splice> SELECT SUM(Salary) FROM Salaries;
1
-----
277275362

1 row selected
splice> SELECT SUM(Salary) FROM Salaries JOIN Players ON Salaries.ID=Players.ID WHERE Team='Cards';
1
-----
97007230

1 row selected
splice> SELECT SUM(Salary) FROM Salaries JOIN Players ON Salaries.ID=Players.ID WHERE Team='Giants';
1
-----
180268132

1 row selected
```

Analytic Example

This example computes the running total of salaries, per team, counting only the players who make at least \$5 million in salary.

```
splice> SELECT Team, DisplayName, Salary,
    SUM(Salary) OVER(PARTITION BY Team ORDER BY Salary ASC
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) "Running Total"
    FROM Players JOIN Salaries ON Players.ID=Salaries.ID
    WHERE Salary>5000000
    ORDER BY Team;
```

TEAM	DISPLAYNAME	SALARY	RUNNING TOTAL
Cards	Larry Lintos	7000000	7000000
Cards	Jack Hellman	8300000	15300000
Cards	James Grasser	9375000	24675000
Cards	Yuri Milleton	15200000	39875000
Cards	Mitch Hassleman	17000000	56875000
Giants	Jalen Ardson	6000000	60000000
Giants	Steve Raster	6000000	12000000
Giants	Marcus Bamburger	6950000	18950000
Giants	Mark Briste	8000000	26950000
Giants	Jack Peepers	9000000	35950000
Giants	Alex Paramour	10250000	46200000
Giants	Thomas Hillman	12000000	58200000
Giants	Buddy Painter	17277777	75477777
Giants	Tam Lassiter	18000000	93477777
Giants	Harry Pennello	18500000	111977777
Giants	Martin Cassman	20833333	132811110

16 rows selected

See Also

- » [About Data Types](#)
- » [Window and aggregate functions](#)
- » [AVG function](#)
- » [COUNT function](#)
- » [MAX function](#)
- » [MIN function](#)
- » [OVER clause](#)
- » [Using Window Functions](#) in the *Developer Guide*.

TAN

The TAN function returns the tangent of a specified number.

Syntax

```
TAN ( number )
```

number

A [DOUBLE PRECISION](#) number that specifies the angle, in radians, for which you want the tangent computed.

Results

The data type of the returned value is a [DOUBLE PRECISION](#) number.

If *number* is NULL, the result of the function is NULL.

If *number* is 0, the result of the function is 0.

Example

```
splice> VALUES TAN(84.4);
1
-----
-0.45017764606194366

1 row selected
```

See Also

- » [DOUBLE PRECISION](#) data type
- » [ACOS](#) function
- » [ASIN](#) function
- » [ATAN](#) function
- » [ATAN2](#) function
- » [COS](#) function
- » [COSH](#) function

- » [COT function](#)
- » [DEGREES function](#)
- » [RADIANS function](#)
- » [SIN function](#)
- » [SINH function](#)
- » [TANH function](#)

TANH

The TANH function returns the hyperbolic tangent of a specified number.

Syntax

```
TANH ( number )
```

number

A [DOUBLE PRECISION](#) number that specifies the angle, in radians, for which you want the hyperbolic tangent computed.

Results

The data type of the returned value is a [DOUBLE PRECISION](#) number.

If *number* is NULL, the result of the function is NULL.

If *number* is 0, the result of the function is 0.

Example

```
splice> VALUES TANH(1.234);
1
-----
0.8437356625893302

1 row selected
```

See Also

- » [DOUBLE PRECISION](#) data type
- » [ACOS](#) function
- » [ASIN](#) function
- » [ATAN](#) function
- » [ATAN2](#) function
- » [COS](#) function

- » [COSH function](#)
- » [COT function](#)
- » [DEGREES function](#)
- » [RADIANS function](#)
- » [SIN function](#)
- » [SINH function](#)
- » [TAN function](#)

TIME

The TIME function returns a time from a value.

Syntax

```
TIME ( expression )
```

expression

An expression that can be any of the following:

- » A [TIMESTAMP](#) value
- » A valid string representation of a time or timestamp

Results

The returned result is governed by the following rules:

- » If the argument can be NULL, the result can be NULL; if the argument is NULL, the result is the NULLvalue.
- » If the argument is a time, the result is that time value.
- » If the argument is a timestamp, the result is the time part of the timestamp.
- » If the argument is a string, the result is the time represented by the string.

Syntax

```
TIME ( expression )
```

Example

```
splice> VALUES TIME( CURRENT_TIMESTAMP );
1
-----
18:53:13
1 row selected
```

See Also

- » [TIME data type](#)
- » [TIMESTAMP data type](#)

TIMESTAMP

The `TIMESTAMP` function returns a timestamp from a value or a pair of values.

Syntax

```
TIMESTAMP ( expression1 [, expression2 ] )
```

expression1

If *expression2* is also specified, *expression1* must be a date or a valid string representation of a date.

If only *expression1* is specified, it must be one of the following:

- » A `DATE` value
- » A valid SQL string representation of a timestamp

expression2

(Optional). A time or a valid string representation of a time.

Results

The data type of the result depends on how the input expression(s) were specified:

- » If both *expression1* and *expression2* are specified, the result is a timestamp with the date specified by *expression1* and the time specified by *expression2*. The microsecond part of the timestamp is zero.
- » If only *expression1* is specified and it is a timestamp, the result is that timestamp.
- » If only *expression1* is specified and it is a string, the result is the timestamp represented by that string. If *expression1* is a string of length 14, the timestamp has a microsecond part of zero.

Examples

This example converts date and time strings into a timestamp value:

```
splice> VALUES TIMESTAMP('2015-11-12', '19:02:43');
1
-----
2015-11-12 19:02:43.0
1 row selected
```

This query shows the timestamp version of the birth date of each player born in the final quarter of the year:

```
splice> SELECT TIMESTAMP(BirthDate)
   FROM Players
 WHERE MONTH(BirthDate) > 10
 ORDER BY BirthDate;
1
-----
1980-12-19 00:00:00.0
1983-11-06 00:00:00.0
1983-11-28 00:00:00.0
1983-12-24 00:00:00.0
1984-11-22 00:00:00.0
1985-11-07 00:00:00.0
1985-11-26 00:00:00.0
1985-12-21 00:00:00.0
1986-11-13 00:00:00.0
1986-11-24 00:00:00.0
1986-12-16 00:00:00.0
1987-11-12 00:00:00.0
1987-11-16 00:00:00.0
1987-12-17 00:00:00.0
1988-12-21 00:00:00.0
1989-11-17 00:00:00.0
1991-11-15 00:00:00.0

17 rows selected
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)

- » [TIME data type](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [Working with Dates](#) in the *Developer's Guide*

TIMESTAMPADD

The `TIMESTAMPADD` function adds the value of an interval to a timestamp value and returns the sum as a new timestamp. You can supply a negative interval value to subtract from a timestamp.

Syntax

```
TIMESTAMPADD ( interval, count, timeStamp1 )
```

interval

One of the following timestamp interval constants:

- » `SQL_TSI_FRAC_SECOND`
- » `SQL_TSI_SECOND`
- » `SQL_TSI_MINUTE`
- » `SQL_TSI_HOUR`
- » `SQL_TSI_DAY`
- » `SQL_TSI_WEEK`
- » `SQL_TSI_MONTH`
- » `SQL_TSI_QUARTER`
- » `SQL_TSI_YEAR`

count

An integer specifying the number of times the interval is to be added to the timestamp. Use a negative integer value to subtract.

timeStamp1

The timestamp value to which the count of intervals is added.

NOTE: If you use a `datetime` column inside the `TIMESTAMPADD` function in a `WHERE` clause, the optimizer cannot use indexes on that column. We strongly recommend not doing this!

Results

The `TIMESTAMPADD` function returns a timestamp value that is the result of adding *count intervals* to *timeStamp1*.

Examples

The following example displays the current timestamp, and the timestamp value two months from now:

```
splice> VALUES ( CURRENT_TIMESTAMP, TIMESTAMPADD(SQL_TSI_MONTH, 2, CURRENT_TIMESTAMP));
1 | 2
-----
2015-11-23 13:54:16.728 | 2016-01-23 13:54:16.728
1 row selected
```

See Also

- » [About Data Types](#)
- » [TIMESTAMP data value](#)
- » [HOUR function](#)
- » [MINUTE function](#)
- » [SECOND function](#)
- » [TIMESTAMP function](#)
- » [TIMESTAMPDIFF function](#)
- » [Working with Dates](#) in the *Developer's Guide*

TIMESTAMPDIFF

The TIMESTAMPDIFF function finds the difference between two timestamps, in terms of the specified interval.

Syntax

```
TIMESTAMPDIFF ( interval, timeStamp1, timeStamp2 )
```

interval

One of the following timestamp interval constants:

- » SQL_TSI_FRAC_SECOND
- » SQL_TSI_SECOND
- » SQL_TSI_MINUTE
- » SQL_TSI_HOUR
- » SQL_TSI_DAY
- » SQL_TSI_WEEK
- » SQL_TSI_MONTH
- » SQL_TSI_QUARTER
- » SQL_TSI_YEAR

timeStamp1

The first timestamp value.

timeStamp2

The second timestamp value.

NOTE: If you use a datetime column inside the TIMESTAMPDIFF function in a WHERE clause, the optimizer cannot use indexes on that column. We strongly recommend not doing this!

Results

The TIMESTAMPDIFF function returns an integer value representing the count of intervals between the two timestamp values.

Examples

These examples shows the number of years a player was born after Nov 22, 1963:.

```
splice> SELECT ID, BirthDate, TIMESTAMPDIFF(SQL_TSI_YEAR, Date('11/22/1963'), BirthDate) "YearsSinceJFK"
      FROM Players WHERE ID < 11
      ORDER BY Birthdate;
ID    |BIRTHDATE |YearsSinceJFK
-----
7    |1981-07-02|17
6    |1982-01-05|18
8    |1983-04-13|19
10   |1983-11-06|19
9    |1983-12-24|20
4    |1987-01-21|23
1    |1987-03-27|23
2    |1988-04-20|24
3    |1990-10-30|26
5    |1991-01-15|27
10 rows selected
```

See Also

- » [About Data Types](#)
- » [TIMESTAMP data value](#)
- » [HOUR function](#)
- » [MINUTE function](#)
- » [SECOND function](#)
- » [TIMESTAMP function](#)
- » [TIMESTAMPADD function](#)
- » [*Working with Dates* in the Developer's Guide](#)

TO_CHAR

The TO_CHAR function formats a date value into a string.

Syntax

```
TO_CHAR( dateExpr, format );
```

dateExpr

The date value that you want to format.

format

A string that specifies the format you want applied to the date. You can specify formats such as the following:

yyyy-mm-dd

mm/dd/yyyy

dd.mm.yy

dd-mm-yy

Results

This function returns a string (CHAR) value.

Examples

```
splice> VALUES TO_CHAR(CURRENT_DATE, 'mm/dd/yyyy');
1
```

```
-----  
09/22/2014  
1 row selected
```

```
splice> VALUES TO_CHAR(CURRENT_DATE, 'dd-mm-yyyy');
1
```

```
-----  
22-09-2014  
1 row selected
```

```
splice> VALUES TO_CHAR(CURRENT_DATE, 'dd-mm-yy');
1
```

```
-----  
22-09-14
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)
- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_DATE function](#)
- » [WEEK function](#)
- » [Working with Dates](#) in the *Developer's Guide*

TO_DATE

The TO_DATE function formats a date string according to a formatting specification, and returns a DATE values do not store time components.

Syntax

```
TO_DATE( dateStrExpr, formatStr );
```

dateStrExpr

A string expression that contains a date that is formatted according to the format string.

formatStr

A string that specifies the format you want applied to the dateStr. See the [Date and Time Formats](#) section below for more information about format specification.

Results

The result is always a DATE value.

Date and Time Formats

Splice Machine supports date and time format specifications based on the Java [SimpleDateFormat](#) class.

Date and time value formats are used for both parsing input values and for formatting output values. For example, the format specification yyyy-MM-dd HH:mm:ssZ parses or formats values like 2014-03-02 11:47:44-0800.

The remainder of this topic describes format specifications in these sections:

- » [Pattern Specifications](#) contains a table showing details for all of the pattern letters you can use.
- » [Presentation Types](#) describes how certain pattern letters are interpreted for parsing and/or formatting.
- » [Examples](#) contains a number of examples that will help you understand how to use formats.

Pattern Specifications

You can specify formatting or parsing patterns for date-time values using the pattern letters shown in the following table. Note that pattern letters are typically repeated in a format specification. For example, YYYY or YY. Refer to the next section for specific information about multiple pattern letters in the different [presentation types](#).

Pattern Letter	Meaning	Presentation Type	Example
G	Era designator	<i>Text</i>	BC
Y	Year	<i>Year</i>	2015 -or- 15
YY	Week year	<i>Year</i>	2011 -or- 11
M	Month in year	<i>Month</i>	July -or- Jul -or- 07
w	Week in year	<i>Number</i>	27
W	Week in month	<i>Number</i>	3
D	Day in year	<i>Number</i>	<p>212</p> <p>A common usage error is to mistakenly specify DD for the day field:</p> <ul style="list-style-type: none"> » use dd to specify day of month » use DD to specify the day of the year
d	Day in month	<i>Number</i>	13
F	Day of week in month	<i>Number</i>	2
E	Day name in week	<i>Text</i>	Tuesday -or- Tue
u	Day number of week (1=Monday, 7=Sunday)	<i>Number</i>	4
a	AM / PM marker	<i>Text</i>	PM
H	Hour in day (0 – 23)	<i>Number</i>	23
k	Hour in day (1 – 24)	<i>Number</i>	24
K	Hour in AM/PM (0 – 11)	<i>Number</i>	11

Pattern Letter	Meaning	Presentation Type	Example
h	Hour in AM/PM (1 – 12)	Number	12
m	Minute in hour	Number	33
s	Second in minute	Number	55
S	Millisecond	Number	959
z	Time zone	General time zone	Pacific Standard Time -or- PST -or- GMT-08:00
Z	Time zone	RFC 822 time zone	-0800
X	Time zone	ISO 8601 time zone	-08 -or- -0800 -or- -08:00
'	Escape char for text	Delimiter	
''	Single quote	Literal	'

Presentation Types

How a presentation type is interpreted for certain pattern letters depends on the number of repeated letters in the pattern. In some cases, as noted in the following table, other factors can influence how the pattern is interpreted.

Presentation Type	Description
Text	<p>For formatting, if the number of pattern letters is 4 or more, the full form is used. Otherwise, a short or abbreviated form is used, if available.</p> <p>For parsing, both forms are accepted, independent of the number of pattern letters.</p>
Number	<p>For formatting, the number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount.</p> <p>For parsing, the number of pattern letters is ignored unless it's needed to separate two adjacent fields.</p>

Presentation Type	Description
Year (for Gregorian calendar)	<p>For formatting, if the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it is interpreted as a number.</p> <p>For parsing, if the number of pattern letters is more than 2, the year is interpreted literally, regardless of the number of digits; e.g.:</p> <ul style="list-style-type: none"> » if you use the pattern MM/dd/yyyy, the value 01/11/12 parses to January 11, 12 A.D. » if you use the pattern MM/dd/yy, the value 01/11/12 parses to January 11, 2012. <p>If the number of pattern letters is one or two, (y or yy), the abbreviated year is interpreted as relative to a century; this is done by adjusting dates to be within 80 years before and 20 years after the current date.</p>
Year (other calendar systems)	<p>Calendar-system specific forms are applied.</p> <p>For both formatting and parsing, if the number of pattern letters is 4 or more, a calendar specific long form is used. Otherwise, a calendar specific short or abbreviated form is used.</p>
Month	<p>If the number of pattern letters is 3 or more, the month is interpreted as text; otherwise, it is interpreted as a number.</p>
General time zone	<p>Time zones are interpreted as text if they have names.</p> <p>For time zones representing a GMT offset value, the following syntax is used:</p> <pre>GMT Sign Hours : Minutes</pre> <p>where:</p> <p>Sign is + or -</p> <p>Hours is either Digit or Digit Digit, between 0 and 23.</p> <p>Minutes is Digit Digit and must be between 00 and 59.</p> <p>For parsing, RFC 822 time zones are also accepted.</p>

Presentation Type	Description
<i>RFC 822 time zone</i>	<p>For formatting, use the RFC 822 4-digit time zone format is used:</p> <pre>Sign TwoDigitHours Minutes</pre> <p>TwoDigitHours must be between 00 and 23.</p> <p>For parsing General time zones are also accepted.</p>
<i>ISO 8601 time zone</i>	<p>The number of pattern letters designates the format for both formatting and parsing as follows:</p> <pre>Sign TwoDigitHours Z Sign TwoDigitHours Minutes Z Sign TwoDigitHours : Minutes Z</pre> <p>For formatting:</p> <ul style="list-style-type: none"> » if the offset value from GMT is 0, z value is produced » if the number of pattern letters is 1, any fraction of an hour is ignored <p>For parsing, Z is parsed as the UTC time zone designator. Note that General time zones <i>are not accepted</i>.</p>
<i>Delimiter</i>	Use the single quote to escape characters in text strings.
<i>Literal</i>	<p>You can include literals in your format specification by enclosing the character(s) in single quotes.</p> <p>Note that you must escape single quotes to include them as literals, e.g. use ''T'' to include the literal string 'T'.</p>

Formatting Examples

The following table contains a number of examples of date time formats:

Date and Time Pattern	Result
'yyyy.MM.dd G 'at' HH:mm:ss z'	2001.07.04 AD at 12:08:56 PDT
'EEE, MMM d, ''yy'	Wed, Jul 4, '01
'h:mm a'	12:08 PM

Date and Time Pattern	Result
"hh 'o' 'clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyyy.MMMM.dd GGG hh:mm aaa"	02001.July.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss z"	Wed, 4 Jul 2001 12:08:56 -0700
"yyMMddHHmmssZ"	010704120856-0700
"yyyy-MM-dd'T'HH:mm:ss.SSSZ"	2001-07-04T12:08:56.235-0700
"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"	2001-07-04T12:08:56.235-07:00
"YYYY-'W'ww-u"	2001-W27-3

Examples of Using `TO_DATE`

Here are several simple examples:

```

splice> VALUES TO_DATE('2015-01-01', 'YYYY-MM-dd');
1
-----
2015-01-01
1 row selected

splice> VALUES TO_DATE('01-01-2015', 'MM-dd-YYYY');
1
-----
2015-01-01
1 row selected

splice> VALUES (TO_DATE('01-01-2015', 'MM-dd-YYYY') + 30);
1
-----
2015-01-31
1

splice> VALUES (TO_DATE('2015-126', 'MM-DDD'));
1
-----
2015-05-06
1 row selected

splice> VALUES (TO_DATE('2015-026', 'MM-DDD'));
1
-----
2015-01-26

splice> VALUES (TO_DATE('2015-26', 'MM-DD'));
1
-----
2015-01-26
1 row selected

```

And here is an example that shows two interesting aspects of using `TO_DATE`. In this example, the input includes the literal `T`, which means that the format pattern must delimit that letter with single quotes. Since we're delimiting the entire pattern in single quotes, we then have to escape those marks and specify `' 'T' '` in our parsing pattern.

And because this example specifies a time zone (`Z`) in the parsing pattern but not in the input string, the timezone information is not preserved. In this case, that means that the parsed date is actually a day earlier than intended:

```

splice> VALUES TO_DATE('2013-06-18T01:03:30.000-0800', 'yyyy-MM-dd''T''HH:mm:ss.SSS
Z');
1
-----
2013-06-17

```

The solution is to explicitly include the timezone for your locale in the input string:

```
splice> VALUES TO_DATE('2013-06-18T01:03:30.000-08:00','yyyy-MM-dd''T''HH:mm:ss.SSSZ');
1
-----
2013-06-18
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)
- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [WEEK function](#)
- » [*Working with Dates* in the *Developer's Guide*](#)

TRIM

The TRIM function that takes a character expression and returns that expression with leading and/or trailing pad characters removed. Optional parameters indicate whether leading, or trailing, or both leading and trailing pad characters should be removed, and specify the pad character that is to be removed.

Syntax

```
TRIM( [ trimOperands ] trimSource)
```

trimOperands

```
{ { trimType [trimCharacter] FROM
  | trimCharacter FROM
}
```

trimCharacter

A character expression that specifies which character to trim from the source. If this is specified, it must evaluate to either NULL or to a character string whose length is exactly one. If left unspecified, it defaults to the space character (' ').

trimType

```
{LEADING | TRAILING | BOTH}
```

If this value is not specified, the default value of BOTH is used.

trimSource

The character expression to be trimmed

Results

If either *trimCharacter* or *trimSource* evaluates to NULL, the result of the TRIM function is NULL. Otherwise, the result is defined as follows:

- » If *trimType* is LEADING, the result will be the *trimSource* value with all leading occurrences of *trimCharacter* removed.
- » If *trimType* is TRAILING, the result will be the *trimSource* value with all trailing occurrences of *trimCharacter* removed.
- » If *trimType* is BOTH, the result will be the *trimSource* value with all leading AND trailing occurrences of *trimCharacter* removed.

If *trimSource*'s data type is CHAR or VARCHAR, the return type of the TRIM function will be VARCHAR. Otherwise the return type of the TRIM function will be CLOB.

Examples

```

splice> VALUES TRIM('      Space Case      ');
1
-----
Space Case      --- This is the string 'Space Case'

splice> VALUES TRIM(BOTH ' ' FROM '      Space Case      ');
1
-----
Space Case      --- This is the string 'Space Case'

splice> VALUES TRIM(TRAILING ' ' FROM '      Space Case      ');
1
-----
Space Case      --- This is the string '      Space Case'

splice> VALUES TRIM(CAST NULL AS CHAR(1) FROM '      Space Case      ');
1
-----
NULL

splice> VALUES TRIM('o' FROM 'VooDoo');
1
-----
VooD

-- results in an error because trimCharacter can only be 1 character
splice> VALUES TRIM('Do' FROM 'VooDoo');

```

See Also

- » [About Data Types](#)
- » [Concatenation operator](#)
- » [INITCAP function](#)
- » [INSTR function](#)
- » [LCASE function](#)
- » [LENGTH function](#)
- » [LOCATE function](#)
- » [LTRIM function](#)
- » [REGEX_LIKE operator](#)
- » [REPLACE function](#)

- » [RTRIM function](#)
- » [SUBSTR function](#)
- » [UCASE function](#)

TRUNC or TRUNCATE

This topic describes the TRUNCATE built-in function, which you can use to truncate numeric, date, and timestamp values. You can use the abbreviation TRUNC interchangeably with the full name, TRUNCATE.

Syntax

```
TRUNCATE( number      [, numPlaces]  |
           date        [, truncPoint]  |
           timestamp   [, truncPoint] );
```

number

An integer or decimal number to be truncated.

date

A [DATE](#) value to be truncated.

timestamp

A [TIMESTAMP](#) value to be truncated.

numPlaces

An optional integer value that specifies the number of digits to truncate (made zero) when applying this function to a *number*.

- » If this value is positive, that many of the least significant digits (to the right of the decimal point) are truncated: `truncate(123.456, 2)` returns 123.450.
- » If this value is negative, that many of the least significant digits to the left of the decimal point are truncated: `truncate(123.456, -1)` returns 120.000.
- » If this value is zero, the decimal portion of the number is truncated: `truncate(123.456, 0)` returns 123.000.
- » If this value is not specified, the decimal portion of the number is zero'ed, which means that `truncate(123.456)` returns 123.000.

See the [Truncating Numbers](#) examples below.

truncPoint

An optional string that specifies the point at which to truncate (zero) a date or timestamp value. This can be one of the following values:

YEAR or YR

The year value is retained; other values are set to their minimum values.

MONTH or MON or MO

The year and month values are retained; other values are set to their minimum values.

DAY

The year, month, and day values are retained; other values are set to their minimum values.

HOUR or HR

The year, month, day, and hour values are retained; other values are set to their minimum values.

MINUTE or MIN

The year, month, day, hour, and minute values are retained; other values are set to their minimum values.

SECOND or SEC

The year, month, day, hour, minute, and second values are retained; the milliseconds value is set to 0.

MILLISECOND or MILLI

All of the values, including year, month, day, hour, minute, second, and milliseconds are retained.

The default value, if nothing is specified, is DAY.

Examples

Truncating Numbers

```
splice> VALUES TRUNC(1234.456, 2);
1
-----
1234.450

splice> VALUES TRUNCATE(123.456,-1);
1
-----
120.000

splice> VALUES TRUNCATE(123.456,0);
1
-----
123.000

splice> VALUES TRUNCATE(123.456);
1
-----
123.000

splice> VALUES TRUNC(1234.456, 2);
1
-----
1234.450

splice> VALUES TRUNCATE(123.456,-1);
1
-----
120.000

splice> VALUES TRUNCATE(123.456,0);
1
-----
123.000
1 row selected

splice> VALUES TRUNCATE(123.456);
1
-----
123.000

VALUES TRUNCATE(1234.6789, 1);
-----
12345.6000

VALUES TRUNCATE(12345.6789, 2);
-----
12345.6700

VALUES TRUNCATE(12345.6789, -1);
-----
```

```
12340.0000  
  
VALUES TRUNCATE(12345.6789, 0);  
-----  
12345.0000  
  
VALUES TRUNCATE(12345.6789);  
-----  
12345.0000
```

Truncating Dates

```
VALUES TRUNCATE(DATE('1988-12-26'), 'year');  
-----  
1988-01-01  
  
VALUES TRUNCATE(DATE('1988-12-26'), 'month');  
-----  
1988-12-01  
  
VALUES TRUNCATE(DATE('1988-12-26'), 'day');  
-----  
1988-12-26  
  
VALUES TRUNCATE(DATE('1988-12-26'));  
-----  
1988-12-26  
  
VALUES TRUNCATE(DATE('2011-12-26'), 'MONTH');  
-----  
2011-12-01
```

Truncating Timestamps

```
VALUES TRUNCATE(TIMESTAMP('2000-06-07 17:12:30.0'), 'year');
-----
2000-01-01 00:00:00.0

VALUES TRUNCATE(TIMESTAMP('2000-06-07 17:12:30.0'), 'month');
-----
2000-06-01 00:00:00.0

VALUES TRUNCATE(TIMESTAMP('2000-06-07 17:12:30.0'), 'day');
-----
2000-06-07 00:00:00.0

VALUES TRUNCATE(TIMESTAMP('2000-06-07 17:12:30.0'), 'hour');
-----
2000-06-07 17:00:00.0

VALUES TRUNCATE(TIMESTAMP('2000-06-07 17:12:30.0'), 'minute');
-----
2000-06-07 17:12:00.0

VALUES TRUNCATE(TIMESTAMP('2000-06-07 17:12:30.0'), 'second');
-----
2000-06-07 17:12:30.0

VALUES TRUNCATE(TIMESTAMP('2000-06-07 17:12:30.0'), 'MONTH');
-----
2011-12-01 00:00:00.0

VALUES TRUNCATE(TIMESTAMP('2000-06-07 17:12:30.0'));
-----
2011-12-26 00:00:00.0
```

LN or UPPER

UCASE or UPPER returns a string in which all alphabetic characters in the input character expression have been converted to uppercase.

NOTE: UPPER and UCASE follow the database locale.

Syntax

```
UCASE or UPPER ( CharacterExpression )
```

CharacterExpression

A `LONG VARCHAR` data type, or any built-in type that is implicitly converted to a string (but not a bit expression).

Results

The data type of the result is as follows:

- » If the *CharacterExpression* evaluates to `NULL`, this function returns `NULL`.
- » If the *CharacterExpression* is of type `CHAR`.
- » If the *CharacterExpression* is of type `LONG VARCHAR`.
- » Otherwise, the return type is `VARCHAR`.

The length and maximum length of the returned value are the same as the length and maximum length of the parameter.

Example

To return the names of players, use the following clause:

```
splice> SELECT UCASE(DisplayName)
      FROM Players
     WHERE ID < 11;
1
-----
BUDDY PAINTER
BILL BOPPER
JOHN PURSER
BOB CRANKER
MITCH DUFFER
NORMAN AIKMAN
ALEX PARAMOUR
HARRY PENNELLO
GREG BROWN
JASON MINMAN

10 rows selected
```

See Also

- » [About Data Types](#)
- » [Concatenation operator](#)
- » [INITCAP function](#)
- » [INSTR function](#)
- » [LCASE function](#)
- » [LENGTH function](#)
- » [LOCATE function](#)
- » [LTRIM function](#)
- » [REGEX_LIKE operator](#)
- » [REPLACE function](#)
- » [RTRIM function](#)
- » [SUBSTR function](#)
- » [TRIM function](#)

USER

When used outside stored routines, `CURRENT_USER`, `USER`, and `SESSION_USER` all return the authorization identifier of the user who created the SQL session.

`SESSION_USER` also always returns this value when used within stored routines.

If used within a stored routine created with `EXTERNAL SECURITY DEFINER`, however, `CURRENT_USER` and `USER` return the authorization identifier of the user that owns the schema of the routine. This is usually the creating user, although the database owner could be the creator as well.

For information about definer's and invoker's rights, see `CREATE FUNCTION` statement.

Syntax

USER

Example

```
splice> VALUES USER;
1
-----
SPLICE
```

See Also

- » [CURRENT_USER function](#)
- » [SESSION_USER function](#)
- » [CREATE_FUNCTION statement](#)
- » [CREATE_PROCEDURE statement](#)

VARCHAR

The VARCHAR function returns a varying-length character string representation of a character string.

Character to varchar syntax

```
VARCHAR (CharacterStringExpression )
```

CharacterStringExpression

An expression whose value must be of a character-string data type with a maximum length of 32,672 bytes.

Datetime to varchar syntax

```
VARCHAR (DatetimeExpression )
```

DatetimeExpression

An expression whose value must be of a date, time, or timestamp data type.

Example

The Position column in our Players table is defined as CHAR(2). The following query shows how to access position values as VARCHARS:

```
splice> SELECT VARCHAR(Position)
      FROM Players
     WHERE ID < 11;
1
-----
C
1B
2B
SS
3B
LF
CF
RF
OF
RF

10 rows selected
```

See Also

- » [About Data Types](#)
- » [VARCHAR data type](#)

WEEK

The WEEK function returns an integer value representing the week of the year from a date expression.

Syntax

```
WEEK( dateExpr );
```

dateExpr

The date-time expression from which you wish to extract information.

Results

The returned week number is in the range 1 to 53.

Examples

```
splice> SELECT BirthDate, Week(BirthDate) "BirthWeek"
      FROM Players
     WHERE ID < 15;
BIRTHDATE | BIRTHWEEK
-----
1987-03-27 | 13
1988-04-20 | 16
1990-10-30 | 44
1987-01-21 | 4
1991-01-15 | 3
1982-01-05 | 1
1981-07-02 | 27
1983-04-13 | 15
1983-12-24 | 51
1983-11-06 | 44
1990-06-16 | 24
1977-08-30 | 35
1990-03-22 | 12
1982-10-12 | 41
14 rows selected
```

See Also

» [CURRENT_DATE function](#)

- » [DATE data type](#)
- » [DATE function](#)
- » [DAY function](#)
- » [EXTRACT function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [MONTHNAME function](#)
- » [NEXTDAY function](#)
- » [NOW function](#)
- » [QUARTER function](#)
- » [TIME data type](#)
- » [TIMESTAMP function](#)
- » [TO_CHAR function](#)
- » [TO_DATE function](#)
- » [*Working with Dates* in the *Developer's Guide*](#)

YEAR

The YEAR function returns the year part of a value. The argument must be a date, timestamp, or a valid character string representation of a date or timestamp. The result of the function is an integer between 1 and 9999.

Syntax

```
YEAR ( expression )
```

Usage

If the argument is NULL, the result is the NULL value.

Examples

Get the current date:

```
splice> value(current_date);  
1  
-----  
2014-02-25
```

Now get the current year only:

```
splice> value(year(current_date));  
1  
-----  
2015
```

Now get the year value from 60 days ago:

```
splice> value(year(current_date-60));  
1  
-----  
2014
```

Select all players born in 1985 or 1989:

```
splice> SELECT DisplayName, Team, BirthDate
      FROM Players
     WHERE YEAR(BirthDate) IN (1985, 1989)
     ORDER BY BirthDate;
DISPLAYNAME          | TEAM      | BIRTHDATE
-----|-----|-----
Jeremy Johnson       | Cards    | 1985-03-15
Gary Kosovo          | Giants   | 1985-06-12
Michael Hillson     | Cards    | 1985-11-07
Mitch Canepa         | Cards    | 1985-11-26
Edward Erdman       | Cards    | 1985-12-21
Jeremy Packman      | Giants   | 1989-01-01
Nathan Nickels      | Giants   | 1989-05-04
Ken Straiter        | Cards    | 1989-07-20
Marcus Bamburger    | Giants   | 1989-08-01
George Goomba        | Cards    | 1989-08-08
Jack Hellman         | Cards    | 1989-08-09
Elliot Andrews       | Giants   | 1989-08-21
Henry Socomy         | Giants   | 1989-11-17
13 rows selected
```

See Also

- » [CURRENT_DATE function](#)
- » [DATE function](#)
- » [DAY function](#)
- » [LASTDAY function](#)
- » [MONTH function](#)
- » [MONTH_BETWEEN function](#)
- » [NEXTDAY function](#)
- » [TIMESTAMP function](#)
- » [*Working with Dates* in the *Developer's Guide*](#)

Built-in System Procedures and Functions

This section contains the reference documentation for the Splice Machine Built-in SQL System Procedures and Functions, in the following subsections:

- » [Database Admin Procedures and Functions](#)
- » [Database Property Procedures and Functions](#)
- » [Importing Data Procedures and Functions](#)
- » [Jar File Procedures and Functions](#)
- » [Logging Procedures and Functions](#)
- » [Statements and Stored Procedures System Procedures](#)
- » [Statistics Procedures and Functions](#)
- » [System Status Procedures and Functions](#)
- » [Transaction Procedures and Functions](#)

Database Admin Procedures and Functions

These are the system procedures and functions for administering your database:

Procedure / Function Name	Description
SYSCS_UTIL.SYSCS_BACKUP_DATABASE	<p>Backs up the database to a specified backup directory.</p> <p>This procedure is only available in our <i>On-Premise Database</i> product.</p>
SYSCS_UTIL.SYSCS_CANCEL_BACKUP	<p>Cancels a backup.</p> <p>This procedure is only available in our <i>On-Premise Database</i> product.</p>
SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP	<p>Cancels a scheduled daily backup.</p> <p>This procedure is only available in our <i>On-Premise Database</i> product.</p>
SYSCS_UTIL.COMPACT_REGION	Performs a minor compaction on a table or index region.

Procedure / Function Name	Description
<code>SYSCS_UTIL.SYSCS_CREATE_USER</code>	Adds a new user account to a database.
<code>SYSCS_UTIL.SYSCS_DELETE_BACKUP</code>	Delete a specific backup. This procedure is only available in our <i>On-Premise Database</i> product.
<code>SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS</code>	Deletes all backups that were created more than a certain number of days ago. This procedure is only available in our <i>On-Premise Database</i> product.
<code>SYSCS_UTIL.SYSCS_DROP_USER</code>	Removes a user account from a database.
<code>SYSCS_UTIL.GET_ENCODED_REGION_NAME</code>	Returns the encoded name of the HBase region that contains the specified, unencoded Splice Machine table primary key or index values.
<code>SYSCS_UTIL.GET_REGIONS</code>	Retrieves the list of regions containing a range of key values.
<code>SYSCS_UTIL.GET_RUNNING_OPERATIONS</code>	Displays information about each Splice Machine operations running on a server.
<code>SYSCS_UTIL.GET_START_KEY</code>	Retrieves the unencoded start key for a specified HBase table or index region.
<code>SYSCS_UTIL.KILL_OPERATION</code>	Terminates a Splice Machine operation running on the server to which you are connected.
<code>SYSCS_UTIL.MAJOR_COMPACT_REGION</code>	Performs a major compaction on a table or index region.
<code>SYSCS_UTIL.MERGE_REGIONS</code>	Merges two adjacent table or index regions.
<code>SYSCS_UTIL.SYSCS_MODIFY_PASSWORD</code>	Called by a user to change that user's own password.
<code>SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA</code>	Performs a major compaction on a schema
<code>SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE</code>	Performs a major compaction on a table.

Procedure / Function Name	Description
SYSCS_UTIL.SYSCS_REFRESH_EXTERNAL_TABLE	Refreshes the schema of an external table in Splice Machine; use this when the schema of the table's source file has been modified outside of Splice Machine.
SYSCS_UTIL.SYSCS_RESET_PASSWORD	Resets a password that has expired or has been forgotten.
SYSCS_UTIL.SYSCS_RESTORE_DATABASE	Restores a database from a previous backup. This procedure is only available in our <i>On-Premise Database</i> product.
SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP	Schedules a full or incremental database backup to run at a specified time daily. This procedure is only available in our <i>On-Premise Database</i> product.
SYSCS_UTIL.SYSCS_UPDATE_SCHEMA_OWNER	Changes the owner of a schema.
SYSCS_UTIL.VACUUM	Performs clean-up operations on the system.

Database Properties Procedures and Functions

These are the system procedures and functions for working with your database properties:

Procedure / Function Name	Description
SYSCS_UTIL.SYSCS_GET_ALL_PROPERTIES	Displays all of the Splice Machine Derby properties.
SYSCS_UTIL.SYSCS_GET_GLOBAL_DATABASE_PROPERTY function	Fetches the value of the specified property of the database.
SYSCS_UTIL.SYSCS_GET_SCHEMA_INFO	Displays table information for all user schemas, including the HBase regions occupied and their store file size.

Procedure / Function Name	Description
SYSCS_UTIL.SYSCS_PEEK_AT_SEQUENCE function	Allows users to observe the instantaneous current value of a sequence generator without having to query the SYSSEQUENCES system table.
SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY	Sets or deletes the value of a property of the database.

Importing Data Procedures and Functions

These are the system procedures and functions for importing data into your database:

Procedure / Function Name	Description
SYSCS_UTIL.BULK_IMPORT_HFILE	Imports data from an HFile.
SYSCS_UTIL.COMPUTE_SPLIT_KEY	Computes split keys for a table or index.
SYSCS_UTIL.DELETE_SNAPSHOT	Deletes a stored snapshot.
SYSCS_UTIL.IMPORT_DATA	Imports data to a subset of columns in a table.
SYSCS_UTIL.SYSCS_MERGE_DATA_FROM_FILE	Imports data from external files, inserting new records and updating existing records.
SYSCS_UTIL.SET_PURGE_DELETED_ROWS	Enables (or disables) physical deletion of logically deleted rows from a specific table.
SYSCS_UTIL.RESTORE_SNAPSHOT	Restores a table or schema from a stored snapshot.
SYSCS_UTIL.SNAPSHOT_SCHEMA	Creates a Splice Machine snapshot of a schema.
SYSCS_UTIL.SNAPSHOT_TABLE	Creates a Splice Machine snapshot of a specific table.
SYSCS_UTIL.SPLIT_TABLE_OR_INDEX_AT_POINTS	Sets up a table or index in your database with split keys computed by the COMPUTE_SPLIT_KEY procedure.
SYSCS_UTIL.SPLIT_TABLE_OR_INDEX	Computes split keys for a table or index and then sets up the table or index. This combines the functionality of SYSCS_UTIL.COMPUTE_SPLIT_KEY and SYSCS_UTIL.SPLIT_TABLE_OR_INDEX_AT_POINTS.

Procedure / Function Name	Description
SYSCS_UTIL.SYSCS_UPSERT_DATA_FROM_FILE	Imports data from external files, inserting new records and updating existing records.

Jar File Procedures and Functions

These are the system procedures and functions for working with JAR files:

Procedure / Function Name	Description
SQLJ.INSTALL_JAR	Stores a jar file in a database.
SQLJ.REMOVE_JAR	Removes a jar file from a database.
SQLJ.REPLACE_JAR	Replaces a jar file in a database.

Logging Procedures and Functions

These are the system procedures and functions for working with system logs:

Procedure / Function Name	Description
SYSCS_UTIL.SYSCS_GET_LOGGER_LEVEL	Displays the log level of the specified logger.
SYSCS_UTIL.SYSCS_GET_LOGGERS	Displays the names of all Splice Machine loggers in the system.
SYSCS_UTIL.SYSCS_SET_LOGGER_LEVEL	Changes the log level of the specified logger.

Statement and Stored Procedures System Procedures

These are the system procedures and functions for working with executing statements and stored procedures:

Procedure / Function Name	Description
SYSCS_UTIL.SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE	Removes as many compiled statements (plans) as possible from the database-wide statement cache (across all region servers).

Procedure / Function Name	Description
SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE	Removes as many compiled statements (plans) as possible from the database statement cache on your current region server.
SYSCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS	Invalidates all system prepared statements and forces the query optimizer to create new execution plans.
SYSCS_UTIL.SYSCS_UPDATE_METADATA_STORED_STATEMENTS	Updates the execution plan for stored procedures in your database.
SYSCS_UTIL.SYSCS_UPDATE_ALL_SYSTEM_PROCEDURES	Updates the signatures of all of the system procedures in a database.
SYSCS_UTIL.SYSCS_UPDATE_SYSTEM_PROCEDURE	Updates the stored declaration of a specific system procedure in the data dictionary.

Statistics Procedures and Functions

These are the system procedures and functions for managing database statistics:

Procedure / Function Name	Description
SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS	Collects statistics on a specific schema in your database.
SYSCS_UTIL.DISABLE_COLUMN_STATISTICS	Disables collection of statistics on a specific column in a table.
SYSCS_UTIL.DROP_SCHEMA_STATISTICS	Drops statistics for a specific schema in your database.
SYSCS_UTIL.ENABLE_COLUMN_STATISTICS	Enables collection of statistics on a specific column in a table.

System Status Procedures and Functions

These are the system procedures and functions for monitoring and adjusting system status:

Procedure / Function Name	Description
SYSCS_UTIL.SYSCS_GET_ACTIVE_SERVERS	Displays the number of active servers in the Splice cluster.

Procedure / Function Name	Description
SYSCS_UTIL.SYSCS_GET_REGION_SERVER_STATS_INFO	Displays input and output statistics about the cluster.
SYSCS_UTIL.SYSCS_GET_REQUESTS	Displays information about the number of RPC requests that are coming into Splice Machine.
SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS	Displays information about all Splice Machine operations running on the server to which you are connected.
SYSCS_UTIL.SYSCS_GET_SESSION_INFO	Displays session information, including the hostname and session IDs.
SYSCS_UTIL.SYSCS_GET_VERSION_INFO	Displays the version of Splice Machine installed on each server in your cluster.
SYSCS_UTIL.SYSCS_GET_WRITE_INTAKE_INFO	Displays information about the number of writes coming into Splice Machine.
SYSCS_UTIL.KILL_OPERATION	Terminates a Splice Machine operation running on the server to which you are connected.

Transaction Procedures and Functions

These are the system procedures and functions for working with transactions in your database

Procedure / Function Name	Description
SYSCS_UTIL.SYSCS_GET_CURRENT_TRANSACTION	Displays summary information about the current transaction.



For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](#).

SYSCS_UTIL.SYSCS_BACKUP_DATABASE

The `SYSCS_UTIL.SYSCS_BACKUP_DATABASE` system procedure performs an immediate full or incremental backup of your database to a specified backup directory.

Splice Machine supports both full and incremental backups:

- » A *full backup* backs up all of the files/blocks that constitute your database.
- » An *incremental backup* only stores database files/blocks that have changed since a previous backup.

NOTE: The first time that you run an incremental backup, a full backup is performed. Subsequent runs of the backup will only copy information that has changed since the previous backup.

For more information, see the *Backing Up and Restoring* topic.

Syntax

```
SYSCS_UTIL.SYSCS_BACKUP_DATABASE( VARCHAR backupDir,
                                  VARCHAR(30) backupType );
```

backupDir

Specifies the path to the directory in which you want the backup stored. This can be a local directory if you're using the standalone version of Splice Machine, or a directory in your cluster's file system (HDFS or MapR-FS).

NOTE: You must have permissions set properly to use cloud storage as a backup destination. See *Backing Up to Cloud Storage* for information about setting backup permissions properties.

Relative paths are resolved based on the current user directory. To avoid confusion, we strongly recommend that you use an absolute path when specifying the backup destination.

backupType

Specifies the type of backup that you want performed. This must be one of the following values: `full` or `incremental`; any other value produces an error and the backup is not run.

Note that if you specify '`incremental`', Splice Machine checks the `SYS.SYSBACKUP` table to determine if there already is a backup for the system; if not, Splice Machine will perform a full backup, and subsequent backups will be incremental.

Results

This procedure does not return a result.

Backup Resource Allocation

Splice Machine backup jobs use a Map Reduce job to copy HFiles; this process may hang up if the resources required for the Map Reduce job are not available from Yarn. See the Backup Resource Allocation section of our *Troubleshooting Guide* for specific information about allocation of resources.

Usage Notes

There's a subtle issue with performing a backup when you're using a temporary table in your session: although the temporary table is (correctly) not backed up, the temporary table's entry in the system tables will be backed up. When the backup is restored, the table entries will be restored, but the temporary table will be missing.

There's a simple workaround:

1. Exit your current session, which will automatically delete the temporary table and its system table entries.
2. Start a new session (reconnect to your database).
3. Start your backup job.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

JDBC example

The following example performs an immediate full backup to a subdirectory of the `hdfs://home/backup` directory:

```
CallableStatement cs = conn.prepareCall
  ("CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE(?,?)");
  cs.setString(1, 'hdfs://home/backup');
  cs.setString(2, 'full');
  cs.execute();
  cs.close();
```

SQL Example

Backing up a database may take several minutes, depending on the size of your database and how much of it you're backing up.

The following example runs an immediate incremental backup to the `hdfs://home/backup/` directory:

```
splice> CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE( 'hdfs:///home/backup' , 'incremental' );
Statement executed.
```

The following example runs the same backup and stores it on AWS:

```
splice> CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE( 's3://backup1234' , 'incremental' );
Statement executed.
```

And this example does a full backup to a relative directory (relative to your splicemachine directory) on a standalone version of Splice Machine:

```
splice> CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE( './dbBackups' , 'full' );
Statement executed.
```

See Also

- » [Backing Up and Restoring Databases](#)
- » [SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP](#)
- » [SYSCS_UTIL.SYSCS_DELETE_BACKUP](#)
- » [SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS](#)
- » [SYSCS_UTIL.SYSCS_RESTORE_DATABASE](#)
- » [SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP](#)
- » [SYSBACKUP](#)
- » [SYSBACKUPITEMS](#)
- » [SYSBACKUPJOBS](#)

SYSCS_UTIL.BULK_IMPORT_HFILE

The `SYSCS_UTIL.BULK_IMPORT_HFILE` system procedure imports data into your Splice Machine database by first generating HFiles and then importing those HFiles.

Our HFile data import procedure leverages HBase bulk loading, which allows it to import your data at a faster rate; however, using this procedure instead of our standard `SYSCS_UTIL.IMPORT_DATA` procedure means that **constraint checks are not performed during data importation**.

Selecting an Import Procedure

Splice Machine provides four system procedures for importing data:

- » The `SYSCS_UTIL.IMPORT_DATA` procedure imports each input record into a new record in your database.
- » The `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` procedure updates existing records and adds new records to your database. It only differs from `SYSCS_UTIL.MERGE_DATA_FROM_FILE` in that upserting **overwrites** the generated or default value of a column that *is not specified* in your `insertColumnList` parameter when updating a record.
- » The `SYSCS_UTIL.MERGE_DATA_FROM_FILE` procedure updates existing records and adds new records to your database. It only differs from `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` in that merging **does not overwrite** the generated or default value of a column that *is not specified* in your `insertColumnList` parameter when updating a record.
- » This procedure, `SYSCS_BULK_IMPORT_HFILE`, takes advantage of HBase bulk loading to import table data into your database by temporarily converting the table file that you're importing into HFiles, importing those directly into your database, and then removing the temporary HFiles. This procedure has improved performance for large tables; however, the bulk HFile import requires extra work on your part and lacks constraint checking.

Our Importing Data Tutorial includes a decision tree and brief discussion to help you determine which procedure best meets your needs.

Syntax

```
call SYSCS_UTIL.BULK_IMPORT_HFILE (
    schemaName,
    tableName,
    insertColumnList | null,
    fileName,
    columnDelimiter | null,
    characterDelimiter | null,
    timestampFormat | null,
    dateFormat | null,
    timeFormat | null,
    maxBadRecords,
    badRecordDirectory | null,
    oneLineRecords | null,
    charset | null,
    bulkImportDirectory,
    skipSampling
);

```

NOTE: If you have specified skipSampling=true to indicate that you're computing the splits yourself, as described [below](#), the parameter values that you pass to the `SYSCS_UTIL.COMPUTE_SPLIT_KEY` or `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX` procedures should match the values that you pass to this procedure.

Parameters

The following table summarizes the parameters used by `SYSCS_UTIL.BULK_IMPORT_HFILE` and other Splice Machine data importation procedures. Each parameter name links to a more detailed description in our Importing Data Tutorial.

Category	Parameter	Description	Example Value
Table Info	schemaName	The name of the schema of the table in which to import.	SPLICE
	tableName	The name of the table in which to import	playerTeams
Data Location	insertColumnList	The names, in single quotes, of the columns to import. If this is null, all columns are imported.	'ID, TEAM'

Category	Parameter	Description	Example Value
	<code>fileOrDirectoryName</code>	<p>Either a single file or a directory. If this is a single file, that file is imported; if this is a directory, all of the files in that directory are imported. You can import compressed or uncompressed files.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>The SYSCS_UTIL.MERGE_DATA_FROM_FILE procedure only works with single files; you cannot specify a directory name when calling SYSCS_UTIL.MERGE_DATA_FROM_FILE.</p> <p>On a cluster, the files to be imported MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p> </div>	<code>/data/mydata/mytable.csv</code> <code>'s3a://splice-benchmark-data/flat/TPCH/100/region'</code>
Data Formats	<code>oneLineRecords</code>	A Boolean value that specifies whether (<code>true</code>) each record in the import file is contained in one input line, or (<code>false</code>) if a record can span multiple lines.	<code>true</code>
	<code>charset</code>	The character encoding of the import file. The default value is UTF-8.	<code>null</code>
	<code>columnDelimiter</code>	The character used to separate columns, Specify <code>null</code> if using the comma (,) character as your delimiter.	' '
	<code>characterDelimiter</code>	The character is used to delimit strings in the imported data.	' "'
	<code>timestampFormat</code>	<p>The format of timestamps stored in the file. You can set this to <code>null</code> if there are no time columns in the file, or if the format of any timestamps in the file match the <code>Java.sql.Timestamp</code> default format, which is: "yyyy-MM-dd HH:mm:ss".</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <p>All of the timestamps in the file you are importing must use the same format.</p> </div>	<code>'yyyy-MM-dd HH:mm:ss.SSZ'</code>

Category	Parameter	Description	Example Value
	dateFormat	The format of timestamps stored in the file. You can set this to null if there are no date columns in the file, or if the format of any dates in the file match pattern: "yyyy-MM-dd".	yyyy-MM-dd
	timeFormat	The format of time values stored in the file. You can set this to null if there are no time columns in the file, or if the format of any times in the file match pattern: "HH:mm:ss".	HH:mm:ss
Problem Logging	badRecordsAllowed	The number of rejected (bad) records that are tolerated before the import fails. If this count of rejected records is reached, the import fails, and any successful record imports are rolled back. Specify 0 to indicate that no bad records are tolerated, and specify -1 to indicate that all bad records should be logged and allowed.	25
	badRecordDirectory	<p>The directory in which bad record information is logged. Splice Machine logs information to the <import_file_name>.bad file in this directory; for example, bad records in an input file named foo.csv would be logged to a file named <i>badRecordDirectory</i>/foo.csv.bad.</p> <p>On a cluster, this directory MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p>	'importErrsDir'
Bulk HFile Import	bulkImportDirectory (outputDirectory)	<p>For SYSCS_UTIL.BULK_IMPORT_HFILE, this is the name of the directory into which the generated HFiles are written prior to being imported into your database.</p> <p>For the SYSCS_UTIL.COMPUTE_SPLIT_KEY procedure, where it is named outputDirectory, this parameter specifies the directory into which the split keys are written.</p>	hdfs:///tmp/test_hfile_import/

Category	Parameter	Description	Example Value
	skipSampling	<p>The <code>skipSampling</code> parameter is a Boolean value that specifies how you want the split keys used for the bulk HFile import to be computed. Set to <code>false</code> to have <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> automatically determine splits for you.</p> <p>This parameter is only used with the <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> system procedure.</p>	<code>false</code>

Usage

The `SYSCS_UTIL.BULK_IMPORT_HFILE` procedure needs the data that you're importing split into multiple HFiles before it actually imports the data into your database. You can achieve these splits in three ways:

- » You can call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter to `false`. `SYSCS_UTIL.BULK_IMPORT_HFILE` samples the data to determine the splits, then splits the data into multiple HFiles, and then imports the data.
- » You can split the data into HFiles with the `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX` system procedure, which both computes the keys and performs the splits. You then call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter to `true` to import your data.
- » You can split the data into HFiles by first calling the `SYSCS_UTIL.COMPUTE_SPLIT_KEY` procedure and then calling the `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS` procedure to split the table or index. You then call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter to `true` to import your data.

In all cases, `SYSCS_UTIL.BULK_IMPORT_HFILE` automatically deletes the HFiles after the import process has completed.

The Bulk HFile Import Examples section of our *Importing Data Tutorial* describes how these methods differ and provides examples of using them to import data.

Results

`SYSCS_UTIL.BULK_IMPORT_HFILE` displays a summary of the import process results that looks like this:

rowsImported	failedRows	files	dataSize	failedLog
94	0	1	4720	NONE

Examples

The Importing Data: Bulk HFile Examples topic walks you through several examples of importing data with bulk HFiles.

See Also

- » [SYSCS_UTIL.COMPUTE_SPLIT_KEY](#)
- » [SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX](#)
- » [SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS](#)

SYSCS_UTIL.SYSCS_CANCEL_BACKUP

The SYSCS_UTIL.SYSCS_CANCEL_BACKUP system procedure cancels an in-progress backup.

Syntax

```
SYSCS_UTIL.SYSCS_CANCEL_BACKUP( );
```

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

Example

This cancels the currently running backup:

```
CALL SYSCS_UTIL.SYSCS_CANCEL_BACKUP();
```

See Also

- » [Backing Up and Restoring Databases](#)
- » [SYSCS_UTIL.SYSCS_BACKUP_DATABASE](#)
- » [SYSCS_UTIL.SYSCS_DELETE_BACKUP](#)
- » [SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS](#)
- » [SYSCS_UTIL.SYSCS_RESTORE_DATABASE](#)
- » [SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP](#)

SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP

The SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP system procedure cancels a scheduled backup job.

NOTE: Once you cancel a daily backup, it will no longer be scheduled to run.

Syntax

```
SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP( BIGINT jobId );
```

jobId

A BIGINT value that specifies which scheduled backup job you want to cancel.

To find the *jobId* you want to cancel, see the *Backing Up and Restoring* topic.

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

Example

If necessary, you can first query the **SYSBACKUPJOBS** system table to find the *jobId* of the job you want to cancel.

And then cancel that job; for example:

```
splice> SELECT * FROM SYS.SYSBACKUPJOBS;
JOB_ID      |FILESYSTEM          |TYPE        |HOUR_OF_DAY|BEGI
N_TIMESTAMP
-----
41069       | ~/Documents/splicemachine/dbBackups/ |full        | 18         | 09:4
1:05.455

1 row selected

splice> CALL SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP(41069);
Statement executed.
```

See Also

- » [Backing Up and Restoring Databases](#)
- » [SYSCS_UTIL.SYSCS_BACKUP_DATABASE](#)
- » [SYSCS_UTIL.SYSCS_DELETE_BACKUP](#)
- » [SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS](#)
- » [SYSCS_UTIL.SYSCS_RESTORE_DATABASE](#)
- » [SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP](#)
- » [SYSBACKUP](#)
- » [SYSBACKUPITEMS](#)
- » [SYSBACKUPJOBS](#)

SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS

The `SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS` system procedure collects statistics on a specific schema in your database.

NOTE: Once statistics have been collected for a schema, they are automatically used by the query optimizer.

This procedure collects statistics for every table in the schema. It also collects statistics for the index associated with every table in the schema. For example, if you have :

- » a schema named `mySchema`
- » `mySchema` contains two tables: `myTable1` and `myTable2`
- » `myTable1` has two indices: `myTable1Index1` and `myTable1Index2`

Then `SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS` will collect statistics for `myTable1`, `myTable2`, `myTable1Index1`, and `myTable1Index2`.

Syntax

```
SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS( VARCHAR(128) schema,
                                         BOOLEAN staleOnly)
```

schemaName

Specifies the schema for which you want to collect statistics. Passing a `null` or non-existent schema name generates an error.

staleOnly

A `BOOLEAN` value that specifies:

- » If this is `true`, data is only re-collected for partitions that are known to have out of date statistics.
- » If this is `false`, data is collected on all partitions. **Note** that this can significantly increase the time required to collect statistics, and is typically used only when you are not sure about the current quality of statistics in the entire schema.



The `staleOnly` parameter value is currently ignored, but must be specified in your call to this procedure. Its value is always set to `false` in the system code.

Results

This procedure returns a results table that contains:

- » one row for each table
 - » one row per index for every table and its associated index in the schema

Each row contains the following columns:

Column Name	Type	Contents
schemaName	VARCHAR	The name of the schema.
tableName	VARCHAR	The name of the table.
partition	VARCHAR	The name of the region on which statistics were collected.
rowsCollected	INTEGER	The number of rows of statistics that were collected..
partitionSize	BIGINT	The size of the partition in bytes.

Usage Notes

Collecting statistics on a schema can take some time.

SQL Examples

```
splice> CALL SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS( 'SPLICE', false );
schemaName |tableName   |partition          |rowsCollec&|partiti
onSize
-----
SPICE      |PLAYERS     |splice:1440,,1467393447889.cbc33f4635ade| 76    | 351
5
SPICE      |SALARIES    |splice:1456,,1467393749257.7724e0cb12af3| 76    | 1420
SPICE      |BATTING     |splice:1472,,1467393754889.b34f5da64c36e| 44    | 2257
1
SPICE      |PITCHING    |splice:1488,,1467393760434.35ee9880e5090| 32    | 21212
SPICE      |FIELDING    |splice:1504,,1467393775949.674b34acdb182| 44    | 9876

5 rows selected
```

See Also

- » Data Assignments and Comparisons
 - » **SYSCS_UTIL.ENABLE_COLUMN_STATISTICS**

- » [SYSCS_UTIL.DISABLE_COLUMN_STATISTICS](#)
- » [SYSCS_UTIL.DROP_SCHEMA_STATISTICS](#)
- » [Using Statistics](#) in the *Developer's Guide*

SYSCS_UTIL.COMPACT_REGION

The SYSCS_UTIL.COMPACT_REGION system procedure performs a minor compaction on a table region or an index region.

Region names must be specified in HBase-encoded format. You can retrieve the encoded name for a region by calling the SYSCS_UTIL.GET_ENCODED_REGION_NAME system procedure.

A common reason for calling this procedure is to improve compaction performance by only compacting recent updates in a table. For example, you might confine any updates to regions of the current month, so older regions need not be re-compacted.

Syntax

```
SYSCS_UTIL.COMPACT_REGION( VARCHAR schemaName,
                            VARCHAR tableName,
                            VARCHAR indexName,
                            VARCHAR startKey)
```

schemaName

The name of the schema of the table.

tableName

The name of the table to compact.

indexName

NULL or the name of the index.

Specify the name of the index you want to compact; if you are compacting the table, specify NULL for this parameter.

regionName

The **encoded** HBase name of the region you want compacted. You can call the SYSCS_UTIL.GET_ENCODED_REGION_NAME procedure to look up the region name for an unencoded Splice Machine table or index key.

Usage

You can compact a table region by specifying NULL for the index name. To compact an index region, specify both the table name and the index name.

Region compaction is asynchronous, which means that when you invoke this procedure from the command line, Splice Machine issues a compaction request to HBase, and returns control to you immediately; HBase will determine when to subsequently run the compaction.

Results

This procedure does not return a result.

Examples

The following example will perform a minor compaction on the region with encoded key value 8ffc80e3f8ac3b180441371319ea90e2 for table testTable. The encoded key value is first retrieved by passing the unencoded key value, 1 | 2, into the SYSCS_UTIL.GET_ENCODED_REGION_NAME procedure:

```
splice> CALL SYSCS_UTIL.GET_ENCODED_REGION_NAME('SPICE', 'TESTTABLE', null, '1|2',
'|', null, null, null, null);;
ENCODED_REGION_NAME          | START_KEY
Y                           | END_KEY
-----
-----
8ffc80e3f8ac3b180441371319ea90e2    | \x81\x00\x8
2                           | \x81\x00\x84

1 row selected

splice> CALL SYSCS_UTIL.COMPACT_REGION('SPICE', 'testTable', NULL, '8ffc80e3f8ac3b1
80441371319ea90e2');
Statement executed.
```

And this example performs a minor compaction on the region with encoded index key value ff8f9e54519a31e15f264ba6d2b828a4 for index testIndex on table testTable. The encoded key value is first retrieved by passing the unencoded index key value, 1996-04-12 | 155190 | 21168.23 | 0.04, into the SYSCS_UTIL.GET_ENCODED_REGION_NAME procedure:

```
splice> CALL SYSCS_UTIL.GET_ENCODED_REGION_NAME('SPICE', 'TESTTABLE', 'SHIP_INDE
X','1996-04-12|155190|21168.23|0.04', '|', null, null, null, null);
ENCODED_REGION_NAME          | START_KEY
Y                           | END_KEY
-----
-----
ff8f9e54519a31e15f264ba6d2b828a4    | \xEC\xC1\x15\xAD\xCD\x80\x00\xE1\x06\xEE\x0
0\xE4V& |

1 row selected

splice> CALL SYSCS_UTIL.COMPACT_REGION('SPICE', 'testTable', 'testIndex', 'ff8f9e54
519a31e15f264ba6d2b828a4');
Statement executed.
```

See Also

- » [SYSCS_UTIL.GET_ENCODED_REGION_NAME](#)
- » [SYSCS_UTIL.GET_REGIONS](#)
- » [SYSCS_UTIL.GET_START_KEY](#)
- » [SYSCS_UTIL.MAJOR_COMPACT_REGION](#)
- » [SYSCS_UTIL.MERGE_REGIONS](#)

SYSCS_UTIL.COMPUTE_SPLIT_KEY

Use the `SYSCS_UTIL.COMPUTE_SPLIT_KEY` system procedure to compute the split keys for a table or index prior to calling the `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS` procedure to split the data into HFiles. Once you've done that, call `SYSCS_UTIL.BULK_IMPORT_HFILE` system procedure to import your data in HFile format.

Syntax

```
call SYSCS_UTIL.COMPUTE_SPLIT_KEY (
    schemaName,
    tableName,
    indexName,
    columnList | null,
    fileName,
    columnDelimiter | null,
    characterDelimiter | null,
    timestampFormat | null,
    dateFormat | null,
    timeFormat | null,
    maxBadRecords,
    badRecordDirectory | null,
    oneLineRecords | null,
    charset | null,
    outputDirectory
);

```

Parameters

The following table summarizes the parameters used by `SYSCS_UTIL.COMPUTE_SPLIT_KEY` and other Splice Machine data importation procedures. Each parameter name links to a more detailed description in our Importing Data Tutorial.



The parameter values that you pass into this procedure should match the values that you use when you subsequently call the `SYSCS_UTIL.BULK_IMPORT_HFILE` procedure to perform the import.

Category	Parameter	Description	Example Value
Table Info	schemaName	The name of the schema of the table in which to import.	SPLICE
	tableName	The name of the table in which to import	playerTeams
Data Location	insertColumnList	The names, in single quotes, of the columns to import. If this is null, all columns are imported.	'ID, TEAM'

Category	Parameter	Description	Example Value
	<code>fileOrDirectoryName</code>	<p>Either a single file or a directory. If this is a single file, that file is imported; if this is a directory, all of the files in that directory are imported. You can import compressed or uncompressed files.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>The SYSCS_UTIL.MERGE_DATA_FROM_FILE procedure only works with single files; you cannot specify a directory name when calling SYSCS_UTIL.MERGE_DATA_FROM_FILE.</p> <p>On a cluster, the files to be imported MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p> </div>	<code>/data/mydata/mytable.csv</code> <code>'s3a://splice-benchmark-data/flat/TPCH/100/region'</code>
Data Formats	<code>oneLineRecords</code>	A Boolean value that specifies whether (<code>true</code>) each record in the import file is contained in one input line, or (<code>false</code>) if a record can span multiple lines.	<code>true</code>
	<code>charset</code>	The character encoding of the import file. The default value is UTF-8.	<code>null</code>
	<code>columnDelimiter</code>	The character used to separate columns, Specify <code>null</code> if using the comma (,) character as your delimiter.	' '
	<code>characterDelimiter</code>	The character is used to delimit strings in the imported data.	' "'
	<code>timestampFormat</code>	<p>The format of timestamps stored in the file. You can set this to <code>null</code> if there are no time columns in the file, or if the format of any timestamps in the file match the <code>Java.sql.Timestamp</code> default format, which is: "yyyy-MM-dd HH:mm:ss".</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <p>All of the timestamps in the file you are importing must use the same format.</p> </div>	<code>'yyyy-MM-dd HH:mm:ss.SSZ'</code>

Category	Parameter	Description	Example Value
	dateFormat	The format of timestamps stored in the file. You can set this to null if there are no date columns in the file, or if the format of any dates in the file match pattern: "yyyy-MM-dd".	yyyy-MM-dd
	timeFormat	The format of time values stored in the file. You can set this to null if there are no time columns in the file, or if the format of any times in the file match pattern: "HH:mm:ss".	HH:mm:ss
Problem Logging	badRecordsAllowed	The number of rejected (bad) records that are tolerated before the import fails. If this count of rejected records is reached, the import fails, and any successful record imports are rolled back. Specify 0 to indicate that no bad records are tolerated, and specify -1 to indicate that all bad records should be logged and allowed.	25
	badRecordDirectory	<p>The directory in which bad record information is logged. Splice Machine logs information to the <import_file_name>.bad file in this directory; for example, bad records in an input file named foo.csv would be logged to a file named <i>badRecordDirectory</i>/foo.csv.bad.</p> <p>On a cluster, this directory MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p>	'importErrsDir'
Bulk HFile Import	bulkImportDirectory (outputDirectory)	<p>For SYSCS_UTIL.BULK_IMPORT_HFILE, this is the name of the directory into which the generated HFiles are written prior to being imported into your database.</p> <p>For the SYSCS_UTIL.COMPUTE_SPLIT_KEY procedure, where it is named outputDirectory, this parameter specifies the directory into which the split keys are written.</p>	hdfs:///tmp/test_hfile_import/

Category	Parameter	Description	Example Value
	skipSampling	<p>The <code>skipSampling</code> parameter is a Boolean value that specifies how you want the split keys used for the bulk HFile import to be computed. Set to <code>false</code> to have <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> automatically determine splits for you.</p> <p>This parameter is only used with the <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> system procedure.</p>	<code>false</code>

Usage

The `SYSCS_UTIL.BULK_IMPORT_HFILE` procedure needs the data that you're importing split into multiple HFiles before it actually imports the data into your database. You can achieve these splits in three ways:

- » You can call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter to `false`. `SYSCS_UTIL.BULK_IMPORT_HFILE` samples the data to determine the splits, then splits the data into multiple HFiles, and then imports the data.
- » You can split the data into HFiles with the `[SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX]` procedure, which both computes the keys and performs the splits. You then call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter to `true` to import your data.
- » You can split the data into HFiles by first calling this procedure, `SYSCS_UTIL.COMPUTE_SPLIT_KEY`, and then calling the `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS` procedure to split the table or index. You then call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter to `true` to import your data.

In all cases, `SYSCS_UTIL.BULK_IMPORT_HFILE` automatically deletes the HFiles after the import process has completed.

The Bulk HFile Import Examples section of our *Importing Data Tutorial* describes how these methods differ and provides examples of using them to import data.

Examples

The Importing Data: Bulk HFile Examples topic walks you through several examples of importing data with bulk HFiles.

See Also

- » [SYSCS_UTIL.BULK_IMPORT_HFILE](#)
- » [SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX](#)

» [SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS](#)

SYSICS_UTIL.SYSCS_CREATE_USER

The `SYSICS_UTIL.SYSCS_CREATE_USER` system procedure adds a new user account to a database.

This procedure creates users for use with NATIVE authentication.

If NATIVE authentication is not already turned on when you call this procedure:

- » The first user whose credentials are stored must be the database owner.
- » Calling this procedure will turn on NATIVE authentication the next time the database is booted.



Once you turn on NATIVE authentication with this procedure, it remains turned on permanently. There is no way to turn it off.

Syntax

```
SYSICS_UTIL.SYSCS_CREATE_USER(
    IN userName VARCHAR(128),
    IN password VARCHAR(32672)
)
```

userName

A user name that is case-sensitive if you place the name string in double quotes. This user name is an authorization identifier.

Case sensitivity is very specific with user names: if you specify the user name in single quotes, e.g. 'Fred', the system automatically converts it into all Uppercase.

NOTE: The user name is only case sensitive if you double-quote it inside of the single quotes. For example, ' "Fred" ' is a different user name than 'Fred', because 'Fred' is assumed to be case-insensitive.

password

A case-sensitive password.

Results

When you add a new user, a new schema is automatically created with exactly the same name as the user. For example, here's a sequence of an administrator adding a new user named `fred` and then verifying that the schema named `fred` is now active:

```

splice> CALL SYSCS_UTIL.SYSCS_CREATE_USER('fred', 'fredpassword');
Statement executed.
splice> VALUES(CURRENT SCHEMA);
1
-----
SPICE

1 row selected
splice> SET SCHEMA fred;
0 rows inserted/updated/deleted
splice> VALUES(CURRENT SCHEMA);
1
-----
FRED

1 row selected

```

When the new user's credentials are used to connect to the database, his/her default schema will be that new schema. If you want the new user to have access to data in other schemas, such as the SPLICE schema, an administrator will need to explicitly grant those access privileges.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

JDBC example

Create a user named FRED:

```

CallableStatement cs = conn.prepareCall
    ("CALL SYSCS_UTIL.SYSCS_CREATE_USER(?, ?)");
cs.setString(1, "fred");
cs.setString(2, "fredpassword");
cs.execute();
cs.close();

```

Create a user named FreD:

```

CallableStatement cs = conn.prepareCall
    ("CALL SYSCS_UTIL.SYSCS_CREATE_USER(?, ?)");
cs.setString(1, "\"FreD\"");
cs.setString(2, "fredpassword");
cs.execute();
cs.close();

```

SQL Example

Create a user named FRED:

```
splice> CALL SYSCS_UTIL.SYSCS_CREATE_USER('fred', 'fredpassword');
Statement executed.
```

Create a (case sensitive) user named MrBaseball:

```
CALL SYSCS_UTIL.SYSCS_CREATE_USER('MrBaseball', 'pinchhitter')
Statement executed.
```

See Also

- » [SYSCS_UTIL.SYSCS_DROP_USER](#) built-in system procedure

SYSCS_UTIL.SYSCS_DELETE_BACKUP

The `SYSCS_UTIL.SYSCS_DELETE_BACKUP` system procedure deletes a backup that you previously created using either the `SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP` system procedures.

Syntax

```
SYSCS_UTIL.SYSCS_DELETE_BACKUP( BIGINT backupId );
```

backupId

Specifies the ID of the backup job you want to delete.

To find the *jobId* you want to cancel, see the *Backing Up and Restoring* topic.

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

Example

If necessary, you can first query the `SYS.SYSBACKUP` system table to find the `BACKUP_ID` of the job you want to delete; entries in that table include timestamp information.

And then delete that job:

```
splice> SELECT * FROM SYS.SYSBACKUP;
BACKUP_ID | BEGIN_TIMESTAMP | END_TIMESTAMP | STATUS | FILESYSTEM
M          | SCOPE | INCR&| INCREMENTAL_PARENT_&| BACKUP_ITEM
-----
40975     | 2015-11-25 09:32:53.04 | 2015-11-25 09:33:09.081 | S      | /Users/me/Docu
ments/splicemachine/dbBackups | D      | false|-1           | 93
-----
1 row selected

splice> CALL SYSCS_UTIL.SYSCS_DELETE_BACKUP(40975);
Statement executed.
```

See Also

- » [Backing Up and Restoring Databases](#)
- » [SYSCS_UTIL.SYSCS_BACKUP_DATABASE](#)
- » [SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP](#)
- » [SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS](#)
- » [SYSCS_UTIL.SYSCS_RESTORE_DATABASE](#)
- » [SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP](#)
- » [SYSBACKUP](#)
- » [SYSBACKUPITEMS](#)
- » [SYSBACKUPJOBS](#)

SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS

The `SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS` system procedure deletes any backups that are older than a specified number of days (the *backup window*), retaining only those backups that fit into that window.

Backups can consume a lot of disk space, and thus, we recommend regularly scheduling both the creation of new backups and deletion of outdated backups.

Syntax

```
SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS( INT backupWindow );
```

backupWindow

Specifies the number of days of backups that you want retained. Any backups created more than `backupWindow` days ago are deleted.

See the *Backing Up and Restoring* topic in our *Administrator's Guide* for more information.

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

SQL Example

The following example deletes all database backups that were created more than 30 days ago.

```
splice> CALL SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS(30);
Statement executed.
```

See Also

- » [Backing Up and Restoring Databases](#)
- » [SYSCS_UTIL.SYSCS_BACKUP_DATABASE](#)
- » [SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP](#)

- » [SYSCS_UTIL.SYSCS_DELETE_BACKUP](#)
- » [SYSCS_UTIL.SYSCS_RESTORE_DATABASE](#)
- » [SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP](#)
- » [SYSBACKUP](#)
- » [SYSBACKUPITEMS](#)
- » [SYSBACKUPJOBS](#)

SYSCS_UTIL.DELETE_REGION

The SYSCS_UTIL.DELETE_REGION system procedure deletes a Splice Machine table or index region.

This procedure is intended for use only by expert database administrators. Use of this procedure requires extreme caution: you can easily create data inconsistencies.

Syntax

```
SYSCS_UTIL.DELETE_REGION( VARCHAR schemaName,
                           VARCHAR tableName,
                           VARCHAR indexName,
                           VARCHAR regionName,
                           VARCHAR mergeRegion )
```

schemaName

The name of the schema of the table.

tableName

The name of the table.

indexName

NULL or the name of the index.

Specify the name of the index if you are deleting an index region; if you are a table region, specify NULL for this parameter.

regionName

The **encoded** HBase name of the first of the two regions you want merged. You can call the [SYSCS_UTIL.GET_ENCODED_REGION_NAME](#) procedure to look up the region name for an unencoded Splice Machine table or index key.

mergeRegion

Specify TRUE (case-insensitive) to merge the region after deleting all of its HFiles.

Usage

Before invoking SYSCS_UTIL.DELETE_REGION(),:

- » Check region boundaries of the base table and indexes using the [SYSCS_UTIL.GET_REGIONS](#) procedure
- » Identify the set of regions from their indexes and tables, and make sure the index regions contains indexes to base table regions.

This procedure is intended for use only by expert database administrators. Use of this procedure requires extreme caution and is intended: you can easily create data inconsistencies.

Configuration Parameters

There are several configuration options that you need to be aware of when using `SYSCS_UTIL.DELETE_REGION`:

- » The `hbase.hbck.close.timeout` value specifies the amount of time to wait for a region to close. The default value is 2 minutes.
- » The `hbase.hbck.assign.timeout` value specifies the amount of time to wait for a region to be assigned. The default value is 2 minutes.
- » We recommend setting the value of `hbase.rpc.timeout` to 20 minutes when using this procedure.

Results

This procedure does not display a result.

Example

Here's an example of creating a table and then deleting a table region and an index region from it.

Create a Table and Index, and Split Them

```
splice> create table t(a int, b int, c int, primary key(a,b));
0 rows inserted/updated/deleted
splice> create index ti on t(a);
0 rows inserted/updated/deleted
splice> insert into t values (1,1,1), (2,2,2), (4,4,4),(5,5,5);
4 rows inserted/updated/deleted
splice> call syscs_util.syscs_split_table_or_index_at_points('SPLICE','T',null,'x8
3');
Statement executed.
splice> call syscs_util.syscs_split_table_or_index_at_points('SPLICE','T','TI','x8
3');
Statement executed.
```

Display region information for base table and index

```

splice> call syscs_util.get_regions('SPLICE', 'T', 'TI',null,null,null,null,null,null,nul
l,null);
ENCODED_REGION_NAME | SPLICE_START_KEY | SPLICE_END_KEY | HBASE_START_KEY
| HBASE_END_KEY | NUM_HFILES | SIZE | LAST_MODIFICATION_TIME | REGION_NAME
-----
-----  

02953478d84fcb1a7bb44f3eba0c9036 | { NULL } | { 3 } |  

|\x83 | 1 | 1073 | 2017-12-12 11:12:34.0 | splice:1809,,151310595332  

5.02953478d84fcb1a7bb44f3eba0c9036.  

1c1ee3dd90817576ef1148d91666defa | { 3 } | { NULL } | \x83  

| 1 | 1073 | 2017-12-12 11:12:34.0 | splice:1809,\x83,151310595  

3325.1c1ee3dd90817576ef1148d91666defa.  

2 rows selected
splice> call syscs_util.get_regions('SPLICE', 'T', null,null,null,null,null,null,nul
l,null);
ENCODED_REGION_NAME | SPLICE_START_KEY | SPLICE_END_KEY | HBASE_START_KEY
| HBASE_END_KEY | NUM_HFILES | SIZE | LAST_MODIFICATION_TIME | REGION_NAME
-----
-----  

19c21ae5b0b2767403a8beff3148b646 | { NULL, NULL } | { 3, NULL } |  

|\x83 | 1 | 1045 | 2017-12-12 11:12:08.0 | splice:1792,,151310592782  

4.19c21ae5b0b2767403a8beff3148b646.  

6c8ac07d50cc2e606562dc1949705374 | { 3, NULL } | { NULL, NULL } | \x83  

| 1 | 1045 | 2017-12-12 11:12:08.0 | splice:1792,\x83,151310592  

7824.6c8ac07d50cc2e606562dc1949705374.  

2 rows selected

```

Delete one region from base table and one region from index

```

splice> call syscs_util.delete_region('SPLICE','T',null,'19c21ae5b0b2767403a8beff314
8b646', true);
Statement executed.
splice> call syscs_util.delete_region('SPLICE','T','TI','02953478d84fcb1a7bb44f3eba0
c9036', true);
Statement executed.

```

Verify the results

```

splice> call syscs_util.get_regions('SPLICE', 'T', 'TI',null,null,null,null,null,null,
1,null);
ENCODED_REGION_NAME          |SPLICE_START_KEY|SPLICE_END_KEY|HBASE_START_KEY
|HBASE_END_KEY|NUM_HFILES|SIZE|LAST_MODIFICATION_TIME|REGION_NAME
-----
-----|-----|-----|-----|-----|-----|
5e8d1ffdf5e8aaa4e85a851caf17a2d9|{ NULL }|{ NULL }|splice:1809,,151310605454
|1|1073|2017-12-12 11:14:15.0|7.5e8d1ffdf5e8aaa4e85a851caf17a2d9.

1 row selected
splice> select count(*) from t --splice-properties index=null
> ;
1
-----
2

1 row selected
splice> select count(*) from t --splice-properties index=ti
> ;
1
-----
2

1 row selected

```

See Also

- » [SYSCS_UTIL.GET_START_KEY](#)
- » [SYSCS_UTIL.GET_REGIONS](#)
- » [SYSCS_UTIL.MERGE_REGIONS](#)

SYSCS_UTIL.SYSCS_DELETE_SNAPSHOT

The SYSCS_UTIL.SYSCS_DELETE_SNAPSHOT system procedure deletes a previously created Splice Machine snapshot.

NOTE: Snapshots include both the data and indexes for tables.

For more information, see the [Using Snapshots](#) topic.

Syntax

```
SYSCS_UTIL.SYSCS_DELETE_SNAPSHOT( VARCHAR(128) snapshotName );
```

snapshotName

The name of the snapshot that you are deleting.

Results

This procedure does not return a result.

Example

The following example deletes a snapshot:

```
splice> CALL SYSCS_UTIL.DELETE_SNAPSHOT( 'snap_myschema_070417a' );
Statement executed.
```

SYSCS_UTIL.DISABLE_COLUMN_STATISTICS

The `SYSCS_UTIL.DISABLE_COLUMN_STATISTICS` system procedure disables collection of statistics on a specific table column in your database.

Syntax

```
SYSCS_UTIL.DISABLE_COLUMN_STATISTICS(
    VARCHAR(128) schema,
    VARCHAR(128) table,
    VARCHAR(128) columnName)
```

schemaName

Specifies the schema of the table. Passing a `null` or non-existent schema name generates an error.

tableName

Specifies the table name of the table. The string must exactly match the case of the table name, and the argument of "Fred" will be passed to SQL as the delimited identifier '`'Fred'`'. Passing a `null` or non-existent table name generates an error.

columnName

Specifies the name of the column for which you want statistics disabled. Passing a `null` or non-existent column name generates an error.

Results

This procedure does not return a result.

Usage Notes

Statistics are automatically collected on all columns by default. Attempting to disable statistics collection on a keyed column generates an error.

SQL Examples

```
splice> CALL SYSCS_UTIL.DISABLE_COLUMN_STATISTICS('SPICE', 'Salaries', 'Salary');
Statement executed.
```

See Also

- » [Data Assignments and Comparisons](#)
- » [SYSCS_UTIL.ENABLE_COLUMN_STATISTICS](#)
- » [SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS](#)
- » [SYSCS_UTIL.DROP_SCHEMA_STATISTICS](#)
- » [Using Statistics](#)

SYSCS_UTIL.DROP_SCHEMA_STATISTICS

The SYSCS_UTIL.DROP_SCHEMA_STATISTICS system procedure drops statistics for a specific schema in your database.

This procedure drops statistics for every table in the schema. It also drops statistics for the index associated with every table in the schema. For example, if you have :

- » a schema named mySchema
- » mySchema contains two tables: myTable1 and myTable2
- » myTable1 has two indices: myTable1Index1 and myTable1Index2

Then SYSCS_UTIL.DROP_SCHEMA_STATISTICS will drop statistics for myTable1, myTable2, myTable1Index1, and myTable1Index2.

Syntax

```
SYSCS_UTIL.DROP_SCHEMA_STATISTICS( VARCHAR(128) schema );
```

schemaName

Specifies the schema for which you want to drop statistics. Passing a null or non-existent schema name generates an error.

Results

This procedure does not produce a result.

SQL Examples

```
splice> CALL SYSCS_UTIL.DROP_SCHEMA_STATISTICS('MYSHEMA');
Statement executed.
```

See Also

- » [Data Assignments and Comparisons](#)
- » [SYSCS_UTIL.ENABLE_COLUMN_STATISTICS](#)
- » [SYSCS_UTIL.DISABLE_COLUMN_STATISTICS](#)
- » [SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS](#)
- » [Using Statistics](#)

SYSCS_UTIL.SYSCS_DROP_USER

The SYSCS_UTIL.SYSCS_DROP_USER system procedure removes a user account from a database.

This procedure is used in conjunction with NATIVE authentication..

You are not allowed to remove the user account of the database owner.

If you use this procedure to remove a user account, the schemas and data objects owned by the user remain in the database and can be accessed only by the database owner or by other users who have been granted access to them. If the user is created again, then he or she regains access to the schemas and data objects.

Syntax

```
SYSCS_UTIL.SYSCS_DROP_USER( IN userName VARCHAR(128) )
```

userName

A user name that is case-sensitive if you place the name string in double quotes. This user name is an authorization identifier. If the user name is that of the database owner, an error is raised.

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

JDBC example

Drop a user named FRED:

```
CallableStatement cs = conn.prepareCall
("CALL SYSCS_UTIL.SYSCS_DROP_USER('fred')");
cs.execute();
cs.close();
```

SQL Example

Drop a user named FreD:

```
splice> CALL SYSCS_UTIL.SYSCS_DROP_USER('fred');
Statement executed;
```

See Also

» [SYSCS_UTIL.SYSCS_CREATE_USER](#)

SYSCS_UTIL.SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE

The SYSCS_UTIL.SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE stored procedure removes as many compiled statements (plans) as possible from the database-wide statement cache (across all region servers). This procedure does not remove statements related to currently executing queries or to activations that are about to be garbage collected, so the cache is not guaranteed to be completely empty after it completes.

NOTE: The related procedure

[SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE](#) performs the same operation on a single region server.

Syntax

```
SYSCS_UTIL.SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE()
```

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

JDBC Example

```
CallableStatement cs = conn.prepareCall  
("CALL SYSCS_UTIL.SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE()");  
cs.execute();  
cs.close();
```

SQL Example

```
splice> CALL SYSCS_UTIL.SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE();  
Statement executed.
```

See Also

- » [SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE](#)
- » [SYSCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS](#)

SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE

The SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE stored procedure removes as many compiled statements (plans) as possible from the database statement cache. on your current region server. This procedure does not remove statements related to currently executing queries or to activations that are about to be garbage collected, so the cache is not guaranteed to be completely empty after it completes.

NOTE: The related procedure [SYSCS_UTIL.SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE](#) performs the same operation across the entire cluster.

Syntax

```
SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE()
```

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

JDBC Example

```
CallableStatement cs = conn.prepareCall
    ("CALL SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE()");
cs.execute();
cs.close();
```

SQL Example

```
splice> CALL SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE();
Statement executed.
```

See Also

- » [SYSCS_UTIL.SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE](#)
- » [SYSCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS](#)
- » [SYSCS_UTIL.SYSCS_UPDATE_METADATA_STORED_STATEMENTS](#)

SYSCS_UTIL.ENABLE_COLUMN_STATISTICS

The SYSCS_UTIL.ENABLE_COLUMN_STATISTICS system procedure enables collection of statistics on a specific table column in your database.

Syntax

```
SYSCS_UTIL.ENABLE_COLUMN_STATISTICS(
    VARCHAR(128) schema,
    VARCHAR(128) table,
    VARCHAR(128) columnName)
```

schemaName

Specifies the schema of the table. Passing a `null` or non-existent schema name generates an error.

tableName

Specifies the table name of the table. The string must exactly match the case of the table name, and the argument of "Fred" will be passed to SQL as the delimited identifier 'Fred'. Passing a `null` or non-existent table name generates an error.

columnName

Specifies the name of the column for which you want statistics enabled. Passing a `null` or non-existent column name generates an error.

Results

This procedure does not return a result.

Usage Notes

Here are some **important notes** about collecting column statistics:

- » Statistics can only be collected on columns with data types that can be ordered; numeric types, some CHAR types, some BIT types, and date/time types can be ordered.

You can determine if a data type can be ordered by examining the Comparisons table in the Data Assignments and Comparisons topic: any data type with a `Y` in any column in that table can be ordered, and thus can have statistics collected on it.

- » Statistics are automatically collected on all columns by default.

SQL Examples

```
splice> CALL SYSCS_UTIL.ENABLE_COLUMN_STATISTICS('SPLICE', 'Salaries', 'Salary');
Statement executed.
```

See Also

- » [Data Assignments and Comparisons](#)
- » [SYSCS_UTIL.DISABLE_COLUMN_STATISTICS](#)
- » [SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS](#)
- » [SYSCS_UTIL.DROP_SCHEMA_STATISTICS](#)
- » [Using Statistics](#)

SYSCS_UTIL.SYSCS_ENABLE_ENTERPRISE

The SYSCS_UTIL.SYSCS_ENABLE_ENTERPRISE stored procedure unlocks access to features that are only available in the Enterprise Edition of Splice Machine.

Calling SYSCS_UTIL.SYSCS_ENABLE_ENTERPRISE with a valid license key unlocks access to *Enterprise-only* features in Splice Machine such as backing up and restoring your database. However, to unlock bootstrapped authentication and encryption features such as LDAP and Kerberos, you must also modify your `hbase-site.xml` file and restart Splice Machine.

NOTE: Please see the Upgrading to the Enterprise Edition of Splice Machine topic for more information.

Syntax

```
SYSCS_UTIL.SYSCS_ENABLE_ENTERPRISE( STRING license_key );
```

license_key

The license key you received from Splice Machine.

SQL Example

```
splice> CALL SYSCS_UTIL.SYSCS_ENABLE_ENTERPRISE (<your-license-code>);
Statement executed.
```

Results

This procedure does not return a result; however, if you provide an invalid license key, you'll see an error message displayed:

```
splice> CALL SYSCS_UTIL.SYSCS_ENABLE_ENTERPRISE (<bogus-code>);
Error
-----
ERROR XSRSE: Unable to enable the enterprise Manager. Enterprise services are disabled. Contact your Splice Machine representative to enable.
```

See Also

- » Upgrading to the Enterprise Edition of Splice Machine

SYSCS_UTIL.SYSCS_GET_ACTIVE_SERVERS

The SYSCS_UTIL.SYSCS_GET_ACTIVE_SERVERS system procedure displays the active servers in the Splice cluster.

Syntax

```
SYSCS_UTIL.SYSCS_GET_ACTIVE_SERVERS()
```

Results

The displayed results of calling SYSCS_UTIL.SYSCS_GET_ACTIVE_SERVERS include these values:

Value	Description
HOSTNAME	The host on which the server is running.
PORT	The port on which the server is listening for requests.
STARTCODE	The system identifier for the Region Server.

Example

```
splice> CALL SYSCS_UTIL.SYSCS_GET_ACTIVE_SERVERS();
HOSTNAME | PORT | STARTCODE
-----
localhost | 56412 | 1447433590803
1 row selected
```

SYSCS_UTIL.SYSCS_GET_ALL_PROPERTIES

The SYSCS_UTIL.SYSCS_GET_ALL_PROPERTIES system procedure displays all of the Splice Machine Derby properties.

Syntax

```
SYSCS_UTIL.SYSCS_GET_ALL_PROPERTIES()
```

Results

The displayed results of calling SYSCS_UTIL.SYSCS_GET_ALL_PROPERTIES include these values:

Value	Description
KEY	The property name
VALUE	The property value
TYPE	The property type

Example

```

splice> CALL SYSCS_UTIL.SYSCS_GET_ALL_PROPERTIES();
KEY | VALU
E   | TYPE
-----
derby.authentication.builtin.algorithm | SHA-51
2                                | JVM
derby.authentication.native.create.credentials.da&| tru
e                                | JVM
derby.authentication.provider        | NATIVE:spliceDB:LOCA
L                                | JVM
derby.connection.requireAuthentication | tru
e                                | JVM
derby.database.collation           | UCS_BASI
C                                | DATABASE
derby.database.defaultConnectionMode | fullAcces
s                                | SERVICE
derby.database.propertiesOnly      | fals
e                                | SERVICE
derby.database.sqlAuthorization    | tru
e                                | JVM
derby.engineType
| 2                                | SERVICE
derby.language.logQueryPlan        | fals
e                                | SERVICE
derby.language.logStatementText    | fals
e                                | SERVICE
derby.language.updateSystemProcs   | fals
e                                | JVM
derby.locks.escalationThreshold   | 50
0                                | SERVICE
derby.storage.propertiesId         | 1
6                                | SERVICE
derby.storage.rowLocking          | fals
e                                | SERVICE
derby.stream.error.file           | ./splice-derby.lo
g                                | JVM
splice.authentication
E                                | NATIV
splice.debug.logStatementContext  | fals
e                                | JVM
splice.software.buildtime         | 2015-10-21 13:18 -050
0                                | JVM
splice.software.release           | 1.5.1
splice.software.url               | http://www.splicemachine.co
m                                | JVM
splice.software.versionhash       | fe1b10bda
0                                | JVM

```

22 rows selected

SYSCS_UTIL.SYSCS_GET_CURRENT_TRANSACTION

The SYSCS_UTIL.SYSCS_GET_CURRENT_TRANSACTION system procedure displays summary information about the current transaction.

Syntax

```
SYSCS_UTIL.SYSCS_GET_CURRENT_TRANSACTION()
```

Results

The displayed results of calling SYSCS_UTIL.SYSCS_GET_CURRENT_TRANSACTION include these values:

Value	Description
txnId	The ID of the current transaction

Example

```
splice> CALL SYSCS_UTIL.SYSCS_GET_CURRENT_TRANSACTION();
 txnId
 -----
 2081
 1 row selected
```

SYSCS_UTIL.SYSCS_GET_GLOBAL_DATABASE_PROPERTY Function

The SYSCS_UTIL.SYSCS_GET_GLOBAL_DATABASE_PROPERTY function fetches the value of the specified property of the database.

Syntax

```
VARCHAR(32672) SYSCS_UTIL.SYSCS_GET_GLOBAL_DATABASE_PROPERTY(
    IN Key VARCHAR(128)
)
```

Key

The key for the property whose value you want.

NOTE: An error occurs if *Key* is null.

Results

Returns the value of the property. If the value that was set for the property is invalid, the SYSCS_UTIL.SYSCS_GET_GLOBAL_DATABASE_PROPERTY function returns the invalid value, but Splice Machine uses the default value.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

SQL Example

Retrieve the value of the splicemachine.locks.deadlockTimeout property:

```
splice> VALUES SYSCS_UTIL.SYSCS_GET_GLOBAL_DATABASE_PROPERTY( 'splicemachine.locks.deadlockTimeout' );
1
-----
10
1 row selected
```

See Also

- » [SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY](#)

SYSCS_UTIL.GET_ENCODED_REGION_NAME

The SYSCS_UTIL.GET_ENCODED_REGION_NAME system procedure returns the encoded name of the HBase region that contains the Splice Machine table primary key or index values for the unencodedKey value that you specify.

You can call this procedure to retrieve an encoded HBase region name prior to calling the SYSCS_UTIL.MAJOR_COMPACT_REGION, SYSCS_UTIL.COMPAKCT_REGION, or SYSCS_UTIL.MERGE_REGIONS procedures.

Syntax

```
SYSCS_UTIL.GET_ENCODED_REGION_NAME( VARCHAR schemaName,
                                     VARCHAR tableName,
                                     VARCHAR indexName,
                                     VARCHAR unencodedKey,
                                     VARCHAR columnDelimiter,
                                     VARCHAR characterDelimiter,
                                     VARCHAR timestampFormat,
                                     VARCHAR dateFormat,
                                     VARCHAR timeFormat )
```

schemaName

The name of the schema of the table.

tableName

The name of the table.

indexName

NULL or the name of the index.

Specify NULL to indicate that the unencodedKey is the primary key of the base table; specify an index name to indicate that the `unencodedKey` is an index value.

unencodedKey

For a table, this is a comma-separated-value (CSV) representation of the table's primary key (unencoded). For an index, this is the CSV representation of the index columns, also unencoded.

columnDelimiter

The character used to separate columns in unencodedKey. Specify null if using the comma (,) character as your delimiter.

characterDelimiter

Specifies which character is used to delimit strings in unencodedKey. You can specify null or the empty string to use the default string delimiter, which is the double-quote ("").

If your input contains control characters such as newline characters, make sure that those characters are embedded within delimited strings.

To use the single quote (') character as your string delimiter, you need to escape that character. This means that you specify four quotes (' '' ') as the value of this parameter. This is standard SQL syntax.

NOTE: The [Examples](#) section below contains an example that uses the single quote as the string delimiter character.

timestampFormat

The format of timestamps in `unencodedKey`. You can set this to `null` if there are no time columns in the split key, or if the format of any timestamps in the file match the `Java.sql.Timestamp` default format, which is: “`yyyy-MM-dd HH:mm:ss`”.

See the [About Timestamp Formats](#) section in the [SYSCS_UTIL.IMPORT_DATA](#) topic for more information about timestamps.

dateFormat

The format of datestamps in `unencodedKey`. You can set this to `null` if there are no date columns in the `unencodedKey`, or if the format of any dates in the split key match this pattern: “`yyyy-MM-dd`”.

timeFormat

The format of time values stored in `unencodedKey`. You can set this to `null` if there are no time columns in the file, or if the format of any times in the split key match this pattern: “`HH:mm:ss`”.

Usage

Use this procedure to retrieve the HBase-encoded name of a table or index region in your database. These system procedures required encoded region names as parameter values:

- » [SYSCS_UTIL.COMPACTION_REGION](#)
- » [SYSCS_UTIL.MAJOR_COMPACT_REGION](#)
- » [SYSCS_UTIL.MERGE_REGIONS](#)

Results

The displayed results of calling `SYSCS_UTIL.SYSCS_GET_ENCODED_REGION_NAME` include these values:

Value	Description
<code>ENCODED_REGION_NAME</code>	The HBase-encoded name of the region.
<code>START_KEY</code>	The HBase starting key for the region.
<code>END_KEY</code>	The HBase ending key for the region.

Examples

The following call will retrieve the encoded region name for TESTTABLE for a table row that has key value 1 | 2:

```
splice> CALL SYSCS_UTIL.GET_ENCODED_REGION_NAME(
           'SPLICE', 'TESTTABLE', null, '1|2', '|', null, null, null, null););
ENCODED_REGION_NAME          |START_KEY          |END_KEY
-----
8ffc80e3f8ac3b180441371319ea90e2    |\x81\x00\x82    |\x81\x00\x84
1 row selected
```

This call will retrieve the encoded region name for TESTTABLE for a region that contains index value 1996-04-12,155190,21168.23,0.04:

```
splice> CALL SYSCS_UTIL.GET_ENCODED_REGION_NAME(
           'SPLICE', 'TESTTABLE', 'SHIP_INDEX','1996-04-12|155190|21168.23|0.0
4',
           '|', null, null, null, null);
ENCODED_REGION_NAME          |START_KEY
|END_KEY
-----
ff8f9e54519a31e15f264ba6d2b828a4 |\xEC\xC1\x15\xAD\xCD\x80\x00\xE1\x06\xEE\x00\xE4
V&|
1 row selected
```

See Also

- » [SYSCS_UTIL.COMPACT_REGION](#)
- » [SYSCS_UTIL.GET_START_KEY](#)
- » [SYSCS_UTIL.GET_REGIONS](#)
- » [SYSCS_UTIL.MAJOR_COMPACT_REGION](#)
- » [SYSCS_UTIL.MERGE_REGIONS](#)

SYSCS_UTIL.SYSCS_GET_LOGGERS

The `SYSCS_UTIL.SYSCS_GET_LOGGERS` system procedure displays the names of all Splice Machine loggers in the system. Use this to find loggers of interest, if you want to determine or change their log levels.

NOTE: You can read more about Splice Machine loggers in the [Logging](#) topic.

Syntax

```
SYSCS_UTIL.SYSCS_GET_LOGGERS()
```

Results

The displayed results of calling `SYSCS_UTIL.SYSCS_GET_LOGGERS` include these values:

Value	Description
LOGGERNAME	The name of the logger

Example

Here's the output from a call to `SYSCS_UTIL.SYSCS_GET_LOGGERS`, as of Splice Machine Release 1.5:

```
splice> CALL SYSCS_UTIL.SYSCS_GET_LOGGERS();
SPLICELOGGER
-----
com.splicemachine
com.splicemachine.async.HBaseClient
com.splicemachine.async.QueueingAsyncScanner
com.splicemachine.async.RegionClient
com.splicemachine.async.RegionInfo
com.splicemachine.async.Scanner
com.splicemachine.concurrent.LoggingScheduledThreadPoolExecutor
com.splicemachine.constants.SpliceConstants
com.splicemachine.constants.environment.EnvUtils
com.splicemachine.db
com.splicemachine.db.impl.ast.AssignRSNVisitor
com.splicemachine.db.impl.ast.FindHashJoinColumns
com.splicemachine.db.impl.ast.FixSubqueryColRefs
com.splicemachine.db.impl.ast.JoinConditionVisitor
com.splicemachine.db.impl.ast.JsonTreeBuilderVisitor
com.splicemachine.db.impl.ast.PlanPrinter
com.splicemachine.db.impl.ast.RowLocationColumnVisitor
com.splicemachine.db.impl.ast.SpliceDerbyVisitorAdapter
com.splicemachine.db.impl.jdbc.authentication
com.splicemachine.db.impl.sql.catalog
com.splicemachine.db.impl.sql.catalog.DefaultSystemProcedureGenerator
com.splicemachine.db.impl.sql.compile.subquery.exists.ExistsSubqueryPredicate
com.splicemachine.db.impl.sql.execute.operations
com.splicemachine.db.shared.common.sanity
com.splicemachine.derby.ddl.AsynchronousDDLController
com.splicemachine.derby.ddl.DDLWatchRefresher
com.splicemachine.derby.ddl.DDLZookeeperClient
com.splicemachine.derby.ddl.ZooKeeperDDLWatchChecker
com.splicemachine.derby.ddl.ZookeeperDDLWatcher
com.splicemachine.derby.hbase.AbstractSpliceIndexObserver
com.splicemachine.derby.hbase.AbstractSpliceIndexObserver.Compaction
com.splicemachine.derby.hbase.AbstractSpliceIndexObserver.Split
com.splicemachine.derby.hbase.ActivationSerializer
com.splicemachine.derby.hbase.RollForwardAction
com.splicemachine.derby.hbase.RollForwardTask
com.splicemachine.derby.hbase.ShutdownRegionServerObserver
com.splicemachine.derby.hbase.SpliceBaseIndexEndpoint
com.splicemachine.derby.hbase.SpliceBaseOperationRegionScanner
com.splicemachine.derby.hbase.SpliceDerbyCoprocessor
com.splicemachine.derby.hbase.SpliceDriver
com.splicemachine.derby.hbase.SpliceIndexEndpoint
com.splicemachine.derby.hbase.SpliceIndexObserver
com.splicemachine.derby.hbase.SpliceMasterObserver
com.splicemachine.derby.hbase.SpliceObserverInstructions
com.splicemachine.derby.hbase.SpliceOperationRegionObserver
com.splicemachine.derby.hbase.SpliceOperationRegionScanner
com.splicemachine.derby.hbase.SpliceWriteControl
com.splicemachine.derby.iapi.sql.execute.OperationResultSet
```

```
com.splicemachine.derby.iapi.sql.execute.SpliceNoPutResultSet
com.splicemachine.derby.iapi.sql.execute.SpliceOperationContext
com.splicemachine.derby.impl.SpliceMethod
com.splicemachine.derby.impl.SpliceService
com.splicemachine.derby.impl.db.SpliceDatabase
com.splicemachine.derby.impl.job.coprocessor.CoprocessorTaskScheduler
com.splicemachine.derby.impl.job.operation.SinkTask
com.splicemachine.derby.impl.job.scheduler.BaseJobControl
com.splicemachine.derby.impl.job.scheduler.DistributedJobScheduler
com.splicemachine.derby.impl.job.scheduler.JobControl
com.splicemachine.derby.impl.job.scheduler.RegionTaskControl
com.splicemachine.derby.impl.job.scheduler.TaskCallable
com.splicemachine.derby.impl.job.scheduler.WorkStealingTaskScheduler
com.splicemachine.derby.impl.services.streams.ConfiguredStream
com.splicemachine.derby.impl.spark.SpliceSpark
com.splicemachine.derby.impl.sql.catalog
com.splicemachine.derby.impl.sql.catalog.SpliceDataDictionary
com.splicemachine.derby.impl.sql.catalog.upgrade
com.splicemachine.derby.impl.sql.compile.NestedLoopJoinStrategy
com.splicemachine.derby.impl.sql.depend.SpliceDependencyManager
com.splicemachine.derby.impl.sql.execute.LazyDataValueDescriptor
com.splicemachine.derby.impl.sql.execute.LazyStringDataValueDescriptor
com.splicemachine.derby.impl.sql.execute.LazyTimestampDataValueDescriptor
com.splicemachine.derby.impl.sql.execute.SpliceExecutionFactory
com.splicemachine.derby.impl.sql.execute.SpliceGenericConstantActionFactory
com.splicemachine.derby.impl.sql.execute.SpliceGenericResultSetFactory
com.splicemachine.derby.impl.sql.execute.SpliceRealResultSetStatisticsFactory
com.splicemachine.derby.impl.sql.execute.actions.DeleteConstantOperation
com.splicemachine.derby.impl.sql.execute.actions.TransactionReadTask
com.splicemachine.derby.impl.sql.execute.operations.AnyOperation
com.splicemachine.derby.impl.sql.execute.operations.BroadCastJoinRows
com.splicemachine.derby.impl.sql.execute.operations.BroadcastJoinOperation
com.splicemachine.derby.impl.sql.execute.operations.CachedOperation
com.splicemachine.derby.impl.sql.execute.operations.CallStatementOperation
com.splicemachine.derby.impl.sql.execute.operations.DMLWriteOperation
com.splicemachine.derby.impl.sql.execute.operations.DeleteOperation
com.splicemachine.derby.impl.sql.execute.operations.IndexRowReader
com.splicemachine.derby.impl.sql.execute.operations.IndexRowToBaseRowOperation
com.splicemachine.derby.impl.sql.execute.operations.JoinOperation
com.splicemachine.derby.impl.sql.execute.operations.JoinUtils
com.splicemachine.derby.impl.sql.execute.operations.Joiner
com.splicemachine.derby.impl.sql.execute.operations.MergeSortJoinOperation
com.splicemachine.derby.impl.sql.execute.operations.NoRowsOperation
com.splicemachine.derby.impl.sql.execute.operations.NormalizeOperation
com.splicemachine.derby.impl.sql.execute.operations.OperationTree
com.splicemachine.derby.impl.sql.execute.operations.ProjectRestrictOperation
com.splicemachine.derby.impl.sql.execute.operations.RowOperation
com.splicemachine.derby.impl.sql.execute.operations.ScanOperation
com.splicemachine.derby.impl.sql.execute.operations.SpliceBaseOperation
com.splicemachine.derby.impl.sql.execute.operations.SpliceBaseOperation.close
com.splicemachine.derby.impl.sql.execute.operations.TableScanOperation
```

```
com.splicemachine.derby.impl.sql.execute.operations.UpdateOperation
com.splicemachine.derby.impl.sql.execute.operations.scanner.SITableScanner
com.splicemachine.derby.impl.stats.CachedPhysicalStatsStore
com.splicemachine.derby.impl.stats.HBaseColumnStatisticsStore
com.splicemachine.derby.impl.stats.PartitionStatsStore
com.splicemachine.derby.impl.stats.StatisticsTask
com.splicemachine.derby.impl.stats.StatsConstants
com.splicemachine.derby.impl.storage.AbstractMultiScanProvider
com.splicemachine.derby.impl.storage.AbstractScanProvider
com.splicemachine.derby.impl.storage.BaseHashAwareScanBoundary
com.splicemachine.derby.impl.storage.ClientResultScanner
com.splicemachine.derby.impl.storage.ClientScanProvider
com.splicemachine.derby.impl.storage.DistributedClientScanProvider
com.splicemachine.derby.impl.storage.MeasuredResultScanner
com.splicemachine.derby.impl.storage.MultiScanRowProvider
com.splicemachine.derby.impl.storage.RegionAwareScanner
com.splicemachine.derby.impl.storage.ReopenableScanner
com.splicemachine.derby.impl.storage.RowProviders
com.splicemachine.derby.impl.storage.SingleScanRowProvider
com.splicemachine.derby.impl.store.access.BaseSpliceTransaction
com.splicemachine.derby.impl.store.access.HBaseStore
com.splicemachine.derby.impl.store.access.PropertyConglomerate
com.splicemachine.derby.impl.store.access.SpliceAccessManager
com.splicemachine.derby.impl.store.access.SpliceLockFactory
com.splicemachine.derby.impl.store.access.SpliceTransaction
com.splicemachine.derby.impl.store.access.SpliceTransactionContext
com.splicemachine.derby.impl.store.access.SpliceTransactionFactory
com.splicemachine.derby.impl.store.access.SpliceTransactionManager
com.splicemachine.derby.impl.store.access.SpliceTransactionManagerContext
com.splicemachine.derby.impl.store.access.StatsStoreCostController
com.splicemachine.derby.impl.store.access.base.SpliceConglomerate
com.splicemachine.derby.impl.store.access.base.SpliceController
com.splicemachine.derby.impl.store.access.base.SpliceScan
com.splicemachine.derby.impl.store.access.btree.IndexConglomerate
com.splicemachine.derby.impl.store.access.btree.IndexConglomerateFactory
com.splicemachine.derby.impl.store.access.btree.IndexController
com.splicemachine.derby.impl.store.access.hbase.HBaseConglomerate
com.splicemachine.derby.impl.store.access.hbase.HBaseController
com.splicemachine.derby.impl.temp.TempTable
com.splicemachine.derby.jdbc.SpliceTransactionResourceImpl
com.splicemachine.derby.logging.DerbyOutputLoggerWriter
com.splicemachine.derby.management.StatementManager
com.splicemachine.derby.management.TransactionalSysTableWriter
com.splicemachine.derby.utils.ConglomerateUtils
com.splicemachine.derby.utils.DerbyBytesUtil
com.splicemachine.derby.utils.Scans
com.splicemachine.derby.utils.SpliceAdmin
com.splicemachine.derby.utils.SpliceUtils
com.splicemachine.derby.utils.StatisticsAdmin
com.splicemachine.hbase.AbstractBufferedRegionScanner
com.splicemachine.hbase.BufferedRegionScanner
```

```
com.splicemachine.hbase.HBaseRegionLoads
com.splicemachine.hbase.NoRetryCoprocessorRpcChannel
com.splicemachine.hbase.backup.Backup
com.splicemachine.hbase.backup.BackupHFileCleaner
com.splicemachine.hbase.backup.BackupReporter
com.splicemachine.hbase.backup.BackupSystemProcedures
com.splicemachine.hbase.backup.BackupUtils
com.splicemachine.hbase.backup.SnapshotUtilsBase
com.splicemachine.hbase.backup.SnapshotUtilsImpl
com.splicemachine.hbase.regioninfocache.HBaseRegionCache
com.splicemachine.hbase.table.BetterHTablePool
com.splicemachine.hbase.table.SpliceHTable
com.splicemachine.hbase.table.SpliceHTableFactory
com.splicemachine.job.CompositeJobResults
com.splicemachine.job.ZkTaskMonitor
com.splicemachine.mrio.api
com.splicemachine.pipeline.callbuffer.PipingCallBuffer
com.splicemachine.pipeline.callbuffer.RegionServerCallBuffer
com.splicemachine.pipeline.ddl.DDLChange
com.splicemachine.pipeline.exception.SpliceDoNotRetryIOException
com.splicemachine.pipeline.impl.BulkWriteAction
com.splicemachine.pipeline.impl.BulkWriteAction.retries
com.splicemachine.pipeline.impl.BulkWriteChannelInvoker
com.splicemachine.pipeline.threadpool.MonitoredThreadPool
com.splicemachine.pipeline.utils.PipelineConstants
com.splicemachine.pipeline.utils.PipelineUtils
com.splicemachine.pipeline.writeconfiguration.BaseWriteConfiguration
com.splicemachine.pipeline.writecontext.PipelineWriteContext
com.splicemachine.pipeline.writecontextfactory.LocalWriteContextFactory
com.splicemachine.pipeline.writehandler.RegionWriteHandler
com.splicemachine.queryPlan
com.splicemachine.si.api.Txn
com.splicemachine.si.coprocessors.SIBaseObserver
com.splicemachine.si.coprocessors.SIOObserver
com.splicemachine.si.coprocessors.TimestampMasterObserver
com.splicemachine.si.coprocessors.TxnLifecycleEndpoint
com.splicemachine.si.impl.BaseSIFilter
com.splicemachine.si.impl.ClientTxnLifecycleManager
com.splicemachine.si.impl.PackedTxnFilter
com.splicemachine.si.impl.ReadOnlyTxn
com.splicemachine.si.impl.SITransactor
com.splicemachine.si.impl.WritableTxn
com.splicemachine.si.impl.readresolve.AsyncReadResolver
com.splicemachine.si.impl.readresolve.SynchronousReadResolver
com.splicemachine.si.impl.region.RegionTxnStore
com.splicemachine.si.impl.region.TransactionResolver
com.splicemachine.si.impl.rollforward.SegmentedRollForward
com.splicemachine.si.impl.timestamp.TimestampClient
com.splicemachine.si.impl.timestamp.TimestampOracle
com.splicemachine.tools.version.ManifestFinder
com.splicemachine.utils.SpliceUtilities
```

```
com.splicemachine.utils.SpliceZooKeeperManager  
com.splicemachine.utils.ZkUtils  
  
203 rows selected
```

See Also

- » [SYSCS_UTIL.SYSCS_GET_LOGGER_LEVEL](#)
- » [SYSCS_UTIL.SYSCS_SET_LOGGER_LEVEL](#)

SYSCS_UTIL.SYSCS_GET_LOGGER_LEVEL

The SYSCS_UTIL.SYSCS_GET_LOGGER_LEVEL system procedure displays the logging level of the specified logger.

NOTE: You can read more about Splice Machine loggers and logging levels in the [Logging](#) topic of our *Developer's Guide*.

Syntax

```
SYSCS_UTIL.SYSCS_GET_LOGGER_LEVEL(loggerName)
```

loggerName

A string specifying the name of the logger whose logging level you want to find.

You can find all of the available loggers by using the [SYSCS_UTIL.SYSCS_GET_LOGGERS](#) system procedure.

Results

The displayed results of calling SYSCS_UTIL.SYSCS_GET_LOGGER_LEVEL include these values:

Value	Description
LOGLEVEL	The level of the logger. This is one of the following values, which are described in the Logging topic: <ul style="list-style-type: none"> » TRACE » DEBUG » INFO » WARN » ERROR » FATAL

Example

Here are two examples of using this procedure:

```
splice> CALL SYSCS_UTIL.SYSCS_GET_LOGGER_LEVEL('com.splicemachine.utils.SpliceUtilities');
LOG&
-----
WARN

1 row selected

splice> CALL SYSCS_UTIL.SYSCS_GET_LOGGER_LEVEL('com.splicemachine.mrio.api');
LOGL&
-----
DEBUG

1 row selected
```

See Also

- » [SYSCS_UTIL.SYSCS_GET_LOGGERS](#)
- » [SYSCS_UTIL.SYSCS_SET_LOGGER_LEVEL](#)
- » [*Splice Machine Logging*](#) in our *Developer's Guide*.

SYSCS_UTIL.GET_REGIONS

The SYSCS_UTIL.GET_REGIONS system procedure that retrieves the list of regions containing a range of key values.

Syntax

```
SYSCS_UTIL.GET_REGIONS( VARCHAR schemaName,
                        VARCHAR tableName,
                        VARCHAR indexName,
                        VARCHAR startKey,
                        VARCHAR endKey,
                        VARCHAR columnDelimiter,
                        VARCHAR characterDelimiter,
                        VARCHAR timestampFormat,
                        VARCHAR dateFormat,
                        VARCHAR timeFormat )
```

schemaName

The name of the schema of the table.

tableName

The name of the table.

indexName

NULL or the name of the index.

Specify NULL to indicate that the *startKey* is the primary key of the base table; specify an index name to indicate that the *startKey* is an index value.

startKey

For a table, this is a comma-separated-value (CSV) representation of the primary key value for the start of the regions in which you are interested. For an index, this is the CSV representation of the index columns.

endKey

For a table, this is a comma-separated-value (CSV) representation of the primary key value for the end of the regions in which you are interested. For an index, this is the CSV representation of the index columns.

columnDelimiter

The character used to separate columns in *startKey*. Specify null if using the comma (,) character as your delimiter.

characterDelimiter

Specifies which character is used to delimit strings in *startKey*. You can specify null or the empty string to use the default string delimiter, which is the double-quote ("").

If your input contains control characters such as newline characters, make sure that those characters are embedded within delimited strings.

To use the single quote (') character as your string delimiter, you need to escape that character. This means that you specify four quotes ('' '' ') as the value of this parameter. This is standard SQL syntax.

NOTE: The [Examples](#) section below contains an example that uses the single quote as the string delimiter character.

timestampFormat

The format of timestamps in `startKey`. You can set this to `null` if there are no time columns in the split key, or if the format of any timestamps in the file match the `Java.sql.Timestamp` default format, which is: "yyyy-MM-dd HH:mm:ss".

See the [About Timestamp Formats](#) section in the `SYSCS_UTIL.IMPORT_DATA` topic for more information about timestamps.

dateFormat

The format of datestamps in `startKey`. You can set this to `null` if there are no date columns in the `startKey`, or if the format of any dates in the split key match this pattern: "yyyy-MM-dd".

timeFormat

The format of time values stored in `startKey`. You can set this to `null` if there are no time columns in the file, or if the format of any times in the split key match this pattern: "HH:mm:ss".

Usage

Specify the starting and ending key values, this procedure returns information about all regions that span those key values.

Results

The displayed results of calling `SYSCS_UTIL.SYSCS_GET_REGIONS` include these values:

Value	Description
<code>ENCODED_REGION_NAME</code>	The HBase-encoded name of the region.
<code>SPLICE_START_KEY</code>	The unencoded start key, in CSV format, for the list of regions in which you are interested. This is the value you supplied in the <code>startKey</code> parameter. For example: {1, 2}.
<code>SPLICE_END_KEY</code>	The unencoded end key for the region, in CSV format, for the list of regions in which you are interested. This is the value you supplied in the <code>endKey</code> parameter. For example: {1, 6}.

Value	Description
HBASE_START_KEY	The start key for the region, formatted as shown in the HBase Web UI. For example: \x81\x00\x82.
HBASE_END_KEY	The end key for the region, formatted as shown in the HBase Web UI. For example: \x81\x00\x86.
NUM_HFILES	The number of HBase Files contained in the region.
SIZE	The size, in bytes, of the region.
LAST_MODIFICATION_TIME	The most recent time at which the region was modified.
REGION_NAME	The unencoded name of the region.

Example

The following call returns information about the regions that are in the key range {1,2} to {1,8}:

```
splice> CALL SYSCS_UTIL.GET_REGIONS( 'SPICE','TestTable', null,
                                     '1|2', '1|8', '|',null,null,null,null);
ENCODED_REGION_NAME | SPLICE_START_KEY | SPLICE_END_KEY | HBASE_START_KEY
| HBASE_END_KEY | NUM_HFILES | SIZE | LAST_MODIFICATION_TIME | REGION_NAME
-----
132c824b9e269006a8e0a3fad577bd12 |{ 1, 2} |{ 1, 6} | \x81\x00\x82
|\x81\x00\x86 |1 |1645 |2017-08-17 12:44:15.0 |splice:2944,\x81\x00\x8
2,1502999053574.132c824b9e269006a8e0a3fad577bd12.
2ee995a552ccb75b7172eed27b917cab |{ 1, 6 } |{ 1, 8 } |\x81\x00\x86
|\x81\x00\x88 |1 |1192 |2017-08-17 08:37:56.0 |splice:2944,\x81\x00\x8
6,1502984266749.2ee995a552ccb75b7172eed27b917cab.

2 rows selected
```

See Also

- » [SYSCS_UTIL.COMPACT_REGION](#)
- » [SYSCS_UTIL.GET_ENCODED_REGION_NAME](#)
- » [SYSCS_UTIL.GET_START_KEY](#)
- » [SYSCS_UTIL.MAJOR_COMPACT_REGION](#)
- » [SYSCS_UTIL.MERGE_REGIONS](#)

SYSICS_UTIL.SYSCS_GET_REGION_SERVER_STATS_INFO

The `SYSCS_UTIL.SYSCS_GET_REGION_SERVER_STATS_INFO` system procedure displays input and output statistics about the cluster.

Syntax

SYSICS_UTIL.SYSCS_GET_REGION_SERVER_STATS_INFO()

Results

The displayed results of calling `SYSICS UTIL.SYSCS GET REGION SERVER STATS INFO` include these values:

Value	Description
HOST	The host name (or IP address).
REGIONCOUNT	The number of regions.
STOREFILECOUNT	The number of files stored.
WRITEREQUESTCOUNT	The number of write requests.
READREQUESTCOUNT	The number of read requests.
TOTALREQUESTCOUNT	The total number of requests.

Example

```
splice> CALL SYSCS_UTIL.SYSCS_GET_REGION_SERVER_STATS_INFO();
Host           |regionCount          |storeFileCount      |writeRequestCount   |readRe
questCount    |totalRequestCount
-----
-----  
111.222.3.4  |58                  |0                  |5956                |9969  
7             |20517  
555.666.7.8  |59                  |0                  |1723                |5702  
2             |6253  
1 row selected
```

SYSCS_UTIL.SYSCS_GET_REQUESTS

The SYSCS_UTIL.SYSCS_GET_REQUESTS system procedure displays information about the number of RPC requests that are coming into Splice Machine.

Syntax

```
SYSCS_UTIL.SYSCS_GET_REQUESTS()
```

Results

The displayed results of calling SYSCS_UTIL.SYSCS_GET_REQUESTS include these values:

Value	Description
HOSTNAME	The host name.
PORT	The port receiving requests.
TOTALREQUESTS	The total number of RPC requests on that port.

Example

```
splice> CALL SYSCS_UTIL.SYSCS_GET_REQUESTS();
HOSTNAME | PORT | TOTALREQUESTS
-----
localhost | 55709 | 7296
1 row selected
```

SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS

The SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS system procedure displays a list of the operations running on the server to which you are currently connected.

You can use this procedure to find the UUID for an operation, which you can then use for purposes such as terminating an operation with the [SYSCS_UTIL.SYSCS_KILL_OPERATION](#) system procedure.

Syntax

```
SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS();
```

Results

The displayed results of calling SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS include these values:

Value	Description
UUID	The operation identifier. This is the same identifier that is shown in the Spark console.
USER	The name of the database user.
HOSTNAME	The host on which the server is running.
SESSION	The session ID.
SQL	The SQL statement that is running.
SUBMITTED	The date and time that the operation was submitted.
ELAPSED	Elapsed time since the operation began running.
ENGINE	Which engine (SPARK or CONTROL) is running the operation.
JOBTYPE	The operation type.

Example

```
splice> call SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS();
```

UUID	USER	HOSTNAME	SESSION	SQL
34b0f479-be9a-4933-9b4d-900af218a19c	SPLICE	MacBook-Pro.local:1527	264	select * from sys.systables --splice-properties useSpark=true
4099f016-3c9d-4c62-8059-ff18d3b38a19	SPLICE	MacBook-Pro.local:1527	4	call syscs_util.syscs_get_running_operations()

2 rows selected

```
splice> call SYSCS_UTIL.SYSCS_KILL_OPERATION('4099f016-3c9d-4c62-8059-ff18d3b38a19');
Statement executed.
```

SYSCS_UTIL.SYSCS_GET_SCHEMA_INFO

The SYSCS_UTIL.SYSCS_GET_SCHEMA_INFO system procedure displays table information, including the HBase regions occupied and their store file size, for all user schemas.

Syntax

```
SYSCS_UTIL.SYSCS_GET_SCHEMA_INFO()
```

Results

The displayed results of calling SYSCS_UTIL.SYSCS_GET_SCHEMA_INFO include these values:

Value	Description
SCHEMANAME	The schema to which the table belongs.
TABLENAME	The name of the table. Note that may be more than one row containing a table name; for example, this happens if the table has an index.
ISINDEX	A Boolean value that specifies whether the HBase table is an index table.
HBASEREGIONS	The HBase regions on which the table resides. There can be multiple regions. Each region display shows (tableName, regionId.storeFileSize, memStoreSize, and storeIndexSize MB).

Example

SYSCS_UTIL.SYSCS_GET_SESSION_INFO

The SYSCS_UTIL.SYSCS_GET_SESSION_INFO system procedure displays the hostname and session ID for your current session. You can use this information to correlate your Splice Machine query with a Spark job: the same information is displayed in the Job Id (Job Group) in the Spark console.

Syntax

```
SYSCS_UTIL.SYSCS_GET_SESSION_INFO()
```

Results

The displayed results of calling SYSCS_UTIL.SYSCS_GET_SESSION include these values:

Value	Description
HOSTNAME	The identity of your Splice Machine connection.
SESSION	The ID of your database connection session.

Example

```
splice> CALL SYSCS_UTIL.SYSCS_GET_SESSION_INFO();
HOSTNAME          | SESSION   |
-----
localhost:1527    | 4
1 row selected
```

For this session, you could find your Spark job ID by correlating the displayed host and session IDs with the Job Group information displayed in the Spark console. For example:

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/ Total	Tasks (for all stages): Succeeded/ Total
0 (SPLICE <localhost:1527,4,e5ff1a51-4ac1-4202-a74c-a2d54ab3525a,36096>)	select * from sys.systables --splice- properties useSpark=true (kill) Produce Result Set	2017/08/ 25 14:03:07	8s	0/1	0/1

SYSCS_UTIL.GET_START_KEY

The SYSCS_UTIL.GET_START_KEY system procedure that retrieves retrieves the starting key value, in unencoded format, for a specified HBase region.

Syntax

```
SYSCS_UTIL.GET_START_KEY( VARCHAR schemaName,  
                           VARCHAR tableName,  
                           VARCHAR indexName,  
                           VARCHAR encodedRegionName)
```

schemaName

The name of the schema of the table.

tableName

The name of the table.

indexName

NULL or the name of the index.

Specify NULL to indicate that the startKey is the primary key of the base table; specify an index name to indicate that the startKey is an index value.

encodedRegionName

The HBase-encoded name of the region, which you can retrieve using the [SYSCS_UTIL.GET_ENCODED_REGION_NAME](#) system procedure.

Usage

Use this procedure to discover the starting key value for an HBase region.

Results

Displays the start key for the region in Splice Machine unencoded format.

Example

The following call returns the start key for an HBase table region:

```
splice> CALL SYSCS_UTIL.GET_START_KEY('SPLICE', 'myTable', null, '9d427082bedabb796  
56369b353e401cc');  
START_KEY  
-----  
{ 2, NULL }  
1 row selected
```

The following call returns the start key for for the region that stores index myIndex on table myTable:

```
splice> CALL SYSCS_UTIL.GET_START_KEY('SPLICE', 'myTable', 'myIndex', 'b35fe82916cdd1  
d48bb5c43f60a9b8b5');  
START_KEY  
-----  
{ 1996-04-11, 67310, 45983.16, 0.09 }  
1 row selected
```

See Also

- » [SYSCS_UTIL.COMPACT_REGION](#)
- » [SYSCS_UTIL.GET_ENCODED_REGION_NAME](#)
- » [SYSCS_UTIL.GET_REGIONS](#)
- » [SYSCS_UTIL.MAJOR_COMPACT_REGION](#)
- » [SYSCS_UTIL.MERGE_REGIONS](#)

SYSCS_UTIL.SYSCS_GET_VERSION_INFO

The SYSCS_UTIL.SYSCS_GET_VERSION_INFO system procedure displays the version of Splice Machine installed on each node in your cluster.

Syntax

```
SYSCS_UTIL.SYSCS_GET_VERSION_INFO()
```

Results

This procedure does not return a result.

Example

```
splice> call SYSCS_UTIL.SYSCS_GET_VERSION_INFO();
HOSTNAME          | RELEASE           | IMPLEMENT&| BUILDTIME          | URL
-----
-----  
localhost:52897|2.5.0.1708-SNAPSHOT|85caa07187|2017-02-25 04:56 +0000 |http://www.splicemachine.com  
  
1 row selected
```

SYSCS_UTIL.SYSCS_GET_WRITE_INTAKE_INFO

The SYSCS_UTIL.SYSCS_GET_WRITE_INTAKE_INFO system procedure displays information about the number of writes coming into Splice Machine.

You can use this information to know the number of bulk writes currently active on a server. Each bulk write will contain up to 1000 rows; the compaction and flush queue size, plus the reserved ipc thread setting determine how many writes can execute concurrently.

Syntax

```
SYSCS_UTIL.SYSCS_GET_WRITE_INTAKE_INFO()
```

Results

The displayed results of calling SYSCS_UTIL.SYSCS_GET_WRITE_INTAKE_INFO include these values:

Value	Description
HOSTNAME	The host name.
ACTIVEWITETHREADS	The number of active write threads.
COMPACTONQUEUESIZELIMIT	The compaction queue limit at which writes will be blocked.
FLUSHQUEUELIMIT	The flush queue limit at which writes will be blocked.
IPCRESERVEDPOOL	The number of IPC threads reserved for reads. The maximum number of bulk writes that are allowed currently is equal to the total number of IPC threads minus this value.

Example

```
splice> CALL SYSCS_UTIL.SYSCS_GET_WRITE_INTAKE_INFO();
host          |depThreads |indThreads |depCount    |indCount    |avgThroughpu
t            |oneMinAvgThroughput   |fiveMinAvgThroughput |fifteenMinAvgThroughput|totalRejected
-----
-----
-----
localhost:55709      |0           |0           |0           |0           |0.01145109332258
9732  |5.472367185912236E-4 |0.007057900046373111 |0.0053884511108858845 |0
1 row selected
```

SYSCS_UTIL.IMPORT_DATA

The `SYSCS_UTIL.IMPORT_DATA` system procedure imports data to a new record in a table. You can choose to import all or a subset of the columns from the input data into your database using the `insertColumnList` parameter.

After a successful import completes, a simple report displays, showing how many files were imported, and how many record imports succeeded or failed.

Selecting an Import Procedure

Splice Machine provides four system procedures for importing data:

- » This procedure, `SYSCS_UTIL.IMPORT_DATA`, imports each input record into a new record in your database.
- » The `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` procedure updates existing records and adds new records to your database. It only differs from `SYSCS_UTIL.MERGE_DATA_FROM_FILE` in that upserting **overwrites** the generated or default value of a column that *is not specified* in your `insertColumnList` parameter when updating a record.
- » The `SYSCS_UTIL.MERGE_DATA_FROM_FILE` procedure updates existing records and adds new records to your database. It only differs from `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` in that merging **does not overwrite** the generated or default value of a column that *is not specified* in your `insertColumnList` parameter when updating a record.
- » The `SYSCS_BULK_IMPORT_HFILE` procedure takes advantage of HBase bulk loading to import table data into your database by temporarily converting the table file that you're importing into HFiles, importing those directly into your database, and then removing the temporary HFiles. This procedure has improved performance for large tables; however, the bulk HFile import requires extra work on your part and lacks constraint checking.

Our Importing Data Tutorial includes a decision tree and brief discussion to help you determine which procedure best meets your needs.

Syntax

```
call SYSCS_UTIL.IMPORT_DATA (
    schemaName,
    tableName,
    insertColumnList | null,
    fileOrDirectoryName,
    columnDelimiter | null,
    characterDelimiter | null,
    timestampFormat | null,
    dateFormat | null,
    timeFormat | null,
    badRecordsAllowed,
    badRecordDirectory | null,
    oneLineRecords | null,
    charset | null
);
```

Parameters

The following table summarizes the parameters used by `SYSCS_UTIL.IMPORT_DATA` and other Splice Machine data importation procedures. Each parameter name links to a more detailed description in our Importing Data Tutorial.

Category	Parameter	Description	Example Value
Table Info	schemaName	The name of the schema of the table in which to import.	SPLICE
	tableName	The name of the table in which to import	playerTeams
Data Location	insertColumnList	The names, in single quotes, of the columns to import. If this is <code>null</code> , all columns are imported.	'ID, TEAM'

Category	Parameter	Description	Example Value
	<code>fileOrDirectoryName</code>	<p>Either a single file or a directory. If this is a single file, that file is imported; if this is a directory, all of the files in that directory are imported. You can import compressed or uncompressed files.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>The SYSCS_UTIL.MERGE_DATA_FROM_FILE procedure only works with single files; you cannot specify a directory name when calling SYSCS_UTIL.MERGE_DATA_FROM_FILE.</p> <p>On a cluster, the files to be imported MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p> </div>	<code>/data/mydata/mytable.csv</code> <code>'s3a://splice-benchmark-data/flat/TPCH/100/region'</code>
Data Formats	<code>oneLineRecords</code>	A Boolean value that specifies whether (<code>true</code>) each record in the import file is contained in one input line, or (<code>false</code>) if a record can span multiple lines.	<code>true</code>
	<code>charset</code>	The character encoding of the import file. The default value is UTF-8.	<code>null</code>
	<code>columnDelimiter</code>	The character used to separate columns, Specify <code>null</code> if using the comma (,) character as your delimiter.	' '
	<code>characterDelimiter</code>	The character is used to delimit strings in the imported data.	' "'
	<code>timestampFormat</code>	<p>The format of timestamps stored in the file. You can set this to <code>null</code> if there are no time columns in the file, or if the format of any timestamps in the file match the <code>Java.sql.Timestamp</code> default format, which is: "yyyy-MM-dd HH:mm:ss".</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  All of the timestamps in the file you are importing must use the same format. </div>	<code>'yyyy-MM-dd HH:mm:ss.SSZ'</code>

Category	Parameter	Description	Example Value
	dateFormat	The format of timestamps stored in the file. You can set this to null if there are no date columns in the file, or if the format of any dates in the file match pattern: "yyyy-MM-dd".	yyyy-MM-dd
	timeFormat	The format of time values stored in the file. You can set this to null if there are no time columns in the file, or if the format of any times in the file match pattern: "HH:mm:ss".	HH:mm:ss
Problem Logging	badRecordsAllowed	The number of rejected (bad) records that are tolerated before the import fails. If this count of rejected records is reached, the import fails, and any successful record imports are rolled back. Specify 0 to indicate that no bad records are tolerated, and specify -1 to indicate that all bad records should be logged and allowed.	25
	badRecordDirectory	<p>The directory in which bad record information is logged. Splice Machine logs information to the <import_file_name>.bad file in this directory; for example, bad records in an input file named foo.csv would be logged to a file named <i>badRecordDirectory</i>/foo.csv.bad.</p> <p>On a cluster, this directory MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p>	'importErrsDir'
Bulk HFile Import	bulkImportDirectory (outputDirectory)	<p>For SYSCS_UTIL.BULK_IMPORT_HFILE, this is the name of the directory into which the generated HFiles are written prior to being imported into your database.</p> <p>For the SYSCS_UTIL.COMPUTE_SPLIT_KEY procedure, where it is named outputDirectory, this parameter specifies the directory into which the split keys are written.</p>	hdfs:///tmp/test_hfile_import/

Category	Parameter	Description	Example Value
	skipSampling	<p>The <code>skipSampling</code> parameter is a Boolean value that specifies how you want the split keys used for the bulk HFile import to be computed. Set to <code>false</code> to have <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> automatically determine splits for you.</p> <p>This parameter is only used with the <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> system procedure.</p>	<code>false</code>

Results

`SYSCS_UTIL.IMPORT_DATA` displays a summary of the import process results that looks like this:

rowsImported	failedRows	files	dataSize	failedLog
94	0	1	4720	NONE

This procedure also logs rejected record activity into `.bad` files in the `badRecordDirectory` directory; one file for each imported file.

Importing and Updating Records

What distinguishes `SYSCS_UTIL.IMPORT_DATA` from the similar [SYSCS_UTIL.UPSERT_DATA_FROM_FILE](#) and [SYSCS_UTIL.SYSCS_MERGED_DATA_FROM_FILE](#) procedures is how each works with these specific conditions:

- » You are importing only a subset of data from the input data into your table, either because the table contains less columns than does the input file, or because you've specified a subset of the columns in your `insertColumnList` parameter.
- » Inserting and updating data in a column with generated values.
- » Inserting and updating data in a column with default values.
- » Handling of missing values.

The Importing Data Tutorial: Input Handling topic describes how each of these conditions is handled by the different system procedures.

Record Import Failure Reasons

Typical reasons for a row (record) import to fail include:

- » Improper data expected for a column.
- » Improper number of columns of data.
- » A primary key violation: [SYSCS_UTIL.IMPORT_DATA](#) will only work correctly if the table into which you are inserting/updating has primary keys.

Usage Notes

A few important notes:

- » Splice Machine advises you to run a full compaction (with the [SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE](#) system procedure) after importing large amounts of data into your database.
- » On a cluster, the files to be imported **MUST be on S3, HDFS (or MapR-FS)**, as must the badRecordDirectory directory. If you're using our Database Service product, files can only be imported from S3.

In addition, the files must be readable by the hbase user, and the badRecordDirectory directory must be writable by the hbase user, either by setting the user explicitly, or by opening up the permissions; for example:

```
sudo -su hdfs hadoop fs -chmod 777 /badRecordDirectory
```

Examples

This section presents a couple simple examples.

The Importing Data Usage Examples topic contains a more extensive set of examples.

Example 1: Importing our doc examples player data

This example shows the `IMPORT_DATA` call used to import the Players table into our documentation examples database:

```
splice> CALL SYSCS_UTIL.IMPORT_DATA('SPLICEBALL', 'Players',
  'ID, Team, Name, Position, DisplayName, BirthDate',
  '/Data/DocExamplesDb/Players.csv',
  null, null, null, null, null, 0, null, true, null);
rowsImported | failedRows | files | dataSize | failedLo
g-----
---  

94 | 0 | 1 | 4720 | NONE  

1 row selected
```

Example 2: Specifying a timestamp format for an entire table

Use a single timestamp format for the entire table by explicitly specifying a single `timeStampFormat`.

```
Mike,2013-04-21 09:21:24.98-05
Mike,2013-04-21 09:15:32.78-04
Mike,2013-03-23 09:45:00.68-05
```

You can then import the data with the following call:

```
splice> CALL SYSCS_UTIL.IMPORT_DATA('app','tabx','c1,c2',
  '/path/to/ts3.csv',
  ',', '',
  'yyyy-MM-dd HH:mm:ss.SSZ',
  null, null, 0, null, true, null);
```

Note that for any import use case shown above, the time shown in the imported table depends on the timezone setting in the server timestamp. In other words, given the same csv file, if imported on different servers with timestamps set to different time zones, the value in the table shown will be different. Additionally, daylight savings time may account for a 1-hour difference if timezone is specified.

See Importing Data Usage Examples for more examples.

See Also

- » Our Importing Data Tutorial
- » Importing Data Usage Examples
- » [SYSCS_UTIL.UPSERT_DATA_FROM_FILE](#)
- » [SYSCS_UTIL.MERGE_DATA_FROM_FILE](#)

SQLJ.INSTALL_JAR

The SQLJ.INSTALL_JAR system procedure stores a jar file in a database.

NOTE: For more information about using JAR files, see the [Using Functions and Stored Procedures](#) section in our *Developer's Guide*.

Syntax

```
SQLJ.INSTALL_JAR(IN jar_file_path_or-url VARCHAR(32672),
                  IN qualified_jar_name VARCHAR(32672),
                  IN deploy INTEGER)
```

jar_file_path_or-url

The path or URL of the jar file to add. A path includes both the directory and the file name (unless the file is in the current directory, in which case the directory is optional). For example:

```
d:/todays_build/tours.jar
```

qualified_jar_name

Splice Machine name of the jar file, qualified by the schema name. Two examples:

```
MYSCHEMA.Sample1
-- a delimited identifier
MYSCHEMA."Sample2"
```

deploy

If this set to 1, it indicates the existence of an SQLJ deployment descriptor file. Splice Machine ignores this argument, so it is normally set to 0.

Usage Notes

This procedure will not work properly unless you have first added your procedure to the Derby CLASSPATH variable. For example:

```
CALL SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY('derby.database.classpath', 'SPICE.MY_EXAMPLE_APP');
```

For information about storing and updating stored procedures, and the setting of the Derby classpath, see the [Storing and Updating Splice Machine Functions and Stored Procedures](#) topic.

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

SQL Examples

```
-- Make sure Derby classpath variable is correctly set for our examples
CALL SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY(
    'derby.database.classpath',
    'SPLICE.SAMPLE1_APP:SPLICE.SAMPLE2');

-- install jar from current directory
splice> CALL SQLJ.INSTALL_JAR('tours.jar', 'SPLICE.SAMPLE1_APP', 0);

-- install jar using full path
splice> CALL SQLJ.INSTALL_JAR('c:\myjarfiles\tours.jar', 'SPLICE.SAMPLE1_APP', 0);

-- install jar from remote location
splice> CALL SQLJ.INSTALL_JAR('http://www.example.com/tours.jar', 'SPLICE.SAMPLE2_APP', 0);

-- install jar using a quoted identifier for the
-- Splice Machine jar name
splice> CALL SQLJ.INSTALL_JAR('tours.jar', 'SPLICE."SAMPLE2"', 0);
```

See Also

- » [SQLJ_REMOVE_JAR](#)
- » [SQLJ_REPLACE_JAR](#)

SYSCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS

The SYSCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS system procedure invalidates all system prepared statements, and forces the query optimizer to create new execution plans. You can use this to speed up query execution by the data dictionary when performance has become sub-optimal.

If you notice that `ij show` commands have slowed down, you can call `SYSCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS` to refresh the execution plans.

NOTE: Splice Machine uses prepared statements known as system procedures to access data in the system tables. These procedures are cached, along with their execution plans, in the data dictionary. The cached execution plans can become sub-optimal after you issue a large number of schema-modifying DLL statements, such as defining and/or modifying a number of tables.

Syntax

```
SYSCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS()
```

Results

This procedure does not return a result.

Example

```
splice> CALL SYSCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS();
Statement executed.
```

See Also

- » [SYSCS_UTIL.SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE](#)
- » [SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE](#)

SYSCS_UTIL.SYSCS_KILL_OPERATION

The SYSCS_UTIL.SYSCS_KILL_OPERATION system procedure terminates an operation that is running on the server to which you are currently connected.

You can use the [SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS](#) system procedure. to find the UUID for an operation you want to kill.

Syntax

```
SYSCS_UTIL.SYSCS_KILL_OPERATION(operationId)
```

operationId

The UUID of the operation that you want to terminate.

This is the same UUID that is shown in the Spark console. You can use the

[SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS](#) system procedure to discover the UUID for the operation.

Results

This procedure does not return a result.

Example

```
splice> call SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS();
          |USER      |HOSTNAME        |SESSION    |SQL
-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+
bf610dea-d33e-4304-bf2e-4f10e667aa98 |SPICE     |localhost:1528 |2          |cal
1 SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS()
33567e3c-ef33-46dc-8d10-5ceb79348c2e |SPICE     |localhost:1528 |20         |ins
ert into a select * from a

2 rows selected

splice> call SYSCS_UTIL.SYSCS_KILL_OPERATION('33567e3c-ef33-46dc-8d10-5ceb79348c2
e');
Statement executed.
```

SYSCS_UTIL.MAJOR_COMPACT_REGION

The SYSCS_UTIL.MAJOR_COMPACT_REGION system procedure performs a major compaction on a table region or an index region.

Region names must be specified in HBase-encoded format. You can retrieve the encoded name for a region by calling the [SYSCS_UTIL.GET_ENCODED_REGION_NAME](#) system procedure.

A common reason for calling this procedure is to improve compaction performance by only compacting recent updates in a table. For example, you might confine any updates to regions of the current month, so older regions need not be re-compacted.

Syntax

```
SYSCS_UTIL.MAJOR_COMPACT_REGION( VARCHAR schemaName,
                                 VARCHAR tableName,
                                 VARCHAR indexName,
                                 VARCHAR startKey)
```

schemaName

The name of the schema of the table.

tableName

The name of the table to compact.

indexName

NULL or the name of the index.

Specify the name of the index you want to compact; if you are compacting the table, specify NULL for this parameter.

regionName

The **encoded** HBase name of the region you want compacted. You can call the [SYSCS_UTIL.GET_ENCODED_REGION_NAME](#) procedure to look up the region name for an unencoded Splice Machine table or index key.

Usage

You can compact a table region by specifying NULL for the index name. To compact an index region, specify both the table name and the index name.

Region compaction is asynchronous, which means that when you invoke this procedure from the command line, Splice Machine issues a compaction request to HBase, and returns control to you immediately; HBase will determine when to subsequently run the compaction.

Results

This procedure does not return a result.

Examples

The following example will perform a major compaction on the region with encoded key value 8ffc80e3f8ac3b180441371319ea90e2 for table testTable. The encoded key value is first retrieved by passing the unencoded key value, 1 | 2, into the SYSCS_UTIL.GET_ENCODED_REGION_NAME procedure:

```
splice> CALL SYSCS_UTIL.GET_ENCODED_REGION_NAME('SPICE', 'TESTTABLE',
                                                null, '1|2', '|', null, null, null,
null);;
ENCODED_REGION_NAME | START_KEY | END_KEY
-----
8ffc80e3f8ac3b180441371319ea90e2 | \x81\x00\x82 | \x81\x00\x84
1 row selected

splice> CALL SYSCS_UTIL.COMPACT_REGION('SPICE', 'testTable',
                                         NULL, '8ffc80e3f8ac3b180441371319ea90e2');
Statement executed.
```

And this example performs a major compaction on the region with encoded index key value ff8f9e54519a31e15f264ba6d2b828a4 for index testIndex on table testTable. The encoded key value is first retrieved by passing the unencoded index key value, 1996-04-12 | 155190 | 21168.23 | 0.04, into the SYSCS_UTIL.GET_ENCODED_REGION_NAME procedure:

```
splice> CALL SYSCS_UTIL.GET_ENCODED_REGION_NAME('SPICE', 'TESTTABLE',
                                                'SHIP_INDEX', '1996-04-12|155190|21168.23|0.04',
                                                '|', null, null, null, null);
ENCODED_REGION_NAME | START_KEY | END_KEY
Y
-----
ff8f9e54519a31e15f264ba6d2b828a4 | \xEC\xC1\x15\xAD\xCD\x80\x00\xE1\x06\xEE\x0
0\xE4V& |

1 row selected

splice> CALL SYSCS_UTIL.COMPACT_REGION('SPICE', 'testTable',
                                         'testIndex', 'ff8f9e54519a31e15f264ba6d2b828a
4');
Statement executed.
```

See Also

- » [SYSCS_UTIL.COMPACT_REGION](#)
- » [SYSCS_UTIL.GET_ENCODED_REGION_NAME](#)
- » [SYSCS_UTIL.GET_REGIONS](#)
- » [SYSCS_UTIL.GET_START_KEY](#)
- » [SYSCS_UTIL.MERGE_REGIONS](#)

SYSCS_UTIL.MERGE_DATA_FROM_FILE

The `SYSCS_UTIL.MERGE_DATA_FROM_FILE` system procedure imports data to update an existing record or create a new record in your database. You can choose to import all or a subset of the columns from the input data into your database using the `insertColumnList` parameter.

After a successful import completes, a simple report displays, showing how many files were imported, and how many record imports succeeded or failed.

Selecting an Import Procedure

Splice Machine provides four system procedures for importing data:

- » The `SYSCS_UTIL.IMPORT_DATA` procedure imports each input record into a new record in your database.
- » The `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` procedure updates existing records and adds new records to your database. It only differs from `SYSCS_UTIL.MERGE_DATA_FROM_FILE` in that upserting **overwrites** the generated or default value of a column that *is not specified* in your `insertColumnList` parameter when updating a record.
- » This procedure, `SYSCS_UTIL.MERGE_DATA_FROM_FILE` procedure updates existing records and adds new records to your database. It only differs from `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` in that merging **does not overwrite** the generated or default value of a column that *is not specified* in your `insertColumnList` parameter when updating a record.
- » The `SYSCS_BULK_IMPORT_HFILE` procedure takes advantage of HBase bulk loading to import table data into your database by temporarily converting the table file that you're importing into HFiles, importing those directly into your database, and then removing the temporary HFiles. This procedure has improved performance for large tables; however, the bulk HFile import requires extra work on your part and lacks constraint checking.

Our Importing Data Tutorial includes a decision tree and brief discussion to help you determine which procedure best meets your needs.

Syntax

```
call SYSCS_UTIL.MERGE_DATA_FROM_FILE (
    schemaName,
    tableName,
    insertColumnList | null,
    fileOrDirectoryName,
    columnDelimiter | null,
    characterDelimiter | null,
    timestampFormat | null,
    dateFormat | null,
    timeFormat | null,
    badRecordsAllowed,
    badRecordDirectory | null,
    oneLineRecords | null,
    charset | null
);
```

Parameters

The following table summarizes the parameters used by `SYSCS_UTIL.MERGE_DATA_FROM_FILE` and other Splice Machine data importation procedures. Each parameter name links to a more detailed description in our Importing Data Tutorial.

Category	Parameter	Description	Example Value
Table Info	schemaName	The name of the schema of the table in which to import.	SPLICE
	tableName	The name of the table in which to import	playerTeams
Data Location	insertColumnList	The names, in single quotes, of the columns to import. If this is <code>null</code> , all columns are imported.	'ID, TEAM'

Category	Parameter	Description	Example Value
	<code>fileOrDirectoryName</code>	<p>Either a single file or a directory. If this is a single file, that file is imported; if this is a directory, all of the files in that directory are imported. You can import compressed or uncompressed files.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>The SYSCS_UTIL.MERGE_DATA_FROM_FILE procedure only works with single files; you cannot specify a directory name when calling SYSCS_UTIL.MERGE_DATA_FROM_FILE.</p> <p>On a cluster, the files to be imported MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p> </div>	<code>/data/mydata/mytable.csv</code> <code>'s3a://splice-benchmark-data/flat/TPCH/100/region'</code>
Data Formats	<code>oneLineRecords</code>	A Boolean value that specifies whether (<code>true</code>) each record in the import file is contained in one input line, or (<code>false</code>) if a record can span multiple lines.	<code>true</code>
	<code>charset</code>	The character encoding of the import file. The default value is UTF-8.	<code>null</code>
	<code>columnDelimiter</code>	The character used to separate columns, Specify <code>null</code> if using the comma (,) character as your delimiter.	' '
	<code>characterDelimiter</code>	The character is used to delimit strings in the imported data.	' "'
	<code>timestampFormat</code>	<p>The format of timestamps stored in the file. You can set this to <code>null</code> if there are no time columns in the file, or if the format of any timestamps in the file match the <code>Java.sql.Timestamp</code> default format, which is: "yyyy-MM-dd HH:mm:ss".</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <p>All of the timestamps in the file you are importing must use the same format.</p> </div>	<code>'yyyy-MM-dd HH:mm:ss.SSZ'</code>

Category	Parameter	Description	Example Value
	dateFormat	The format of timestamps stored in the file. You can set this to null if there are no date columns in the file, or if the format of any dates in the file match pattern: "yyyy-MM-dd".	yyyy-MM-dd
	timeFormat	The format of time values stored in the file. You can set this to null if there are no time columns in the file, or if the format of any times in the file match pattern: "HH:mm:ss".	HH:mm:ss
Problem Logging	badRecordsAllowed	The number of rejected (bad) records that are tolerated before the import fails. If this count of rejected records is reached, the import fails, and any successful record imports are rolled back. Specify 0 to indicate that no bad records are tolerated, and specify -1 to indicate that all bad records should be logged and allowed.	25
	badRecordDirectory	<p>The directory in which bad record information is logged. Splice Machine logs information to the <import_file_name>.bad file in this directory; for example, bad records in an input file named foo.csv would be logged to a file named <i>badRecordDirectory</i>/foo.csv.bad.</p> <p>On a cluster, this directory MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p>	'importErrsDir'
Bulk HFile Import	bulkImportDirectory (outputDirectory)	<p>For SYSCS_UTIL.BULK_IMPORT_HFILE, this is the name of the directory into which the generated HFiles are written prior to being imported into your database.</p> <p>For the SYSCS_UTIL.COMPUTE_SPLIT_KEY procedure, where it is named outputDirectory, this parameter specifies the directory into which the split keys are written.</p>	hdfs:///tmp/test_hfile_import/

Category	Parameter	Description	Example Value
	skipSampling	<p>The <code>skipSampling</code> parameter is a Boolean value that specifies how you want the split keys used for the bulk HFile import to be computed. Set to <code>false</code> to have <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> automatically determine splits for you.</p> <p>This parameter is only used with the <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> system procedure.</p>	<code>false</code>

Notes

- » The `SYSCS_UTIL.MERGE_DATA_FROM_FILE` procedure only imports single files; it does not process directories. This means that the `fileOrDirectoryName` parameter value must be a file name.

Results

`SYSCS_UTIL.MERGE_DATA_FROM_FILE` displays a summary of the import process results that looks like this:

rowsImported	failedRows	files	dataSize	failedLog
94	0	1	4720	NONE

This procedure also logs rejected record activity into `.bad` files in the `badRecordDirectory` directory; one file for each imported file.

Importing and Updating Records

What distinguishes `SYSCS_UTIL.IMPORT_DATA` from the similar `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` and `SYSCS_UTIL.SYSCS_MERGED_DATA_FROM_FILE` procedures is how each works with these specific conditions:

- » You are importing only a subset of data from the input data into your table, either because the table contains less columns than does the input file, or because you've specified a subset of the columns in your `insertColumnList` parameter.
- » Inserting and updating data in a column with generated values.
- » Inserting and updating data in a column with default values.
- » Handling of missing values.

The Importing Data Tutorial: Input Handling topic describes how each of these conditions is handled by the different system procedures.

Record Import Failure Reasons

When upserting data from a file, the input file you generate must contain:

- » the columns to be changed
- » all NON_NULL columns

Typical reasons for a row (record) import to fail include:

- » Improper data expected for a column.
- » Improper number of columns of data.
- » A primary key violation: [SYSCS_UTIL.MERGE_DATA_FROM_FILE](#) will only work correctly if the table into which you are inserting/updating has primary keys.

A few important notes:

- » Splice Machine advises you to run a full compaction (with the [SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE](#) system procedure) after importing large amounts of data into your database.
- » On a cluster, the files to be imported **MUST be on S3, HDFS (or MapR-FS)**, as must the badRecordDirectory directory. If you're using our Database Service product, files can only be imported from S3.

In addition, the files must be readable by the hbase user, and the badRecordDirectory directory must be writable by the hbase user, either by setting the user explicitly, or by opening up the permissions; for example:

```
sudo -su hdfs hadoop fs -chmod 777 /badRecordDirectory
```

Examples

This section presents a couple simple examples.

The Importing Data Usage Examples topic contains a more extensive set of examples.

Example 1: Updating our doc examples player data

This example shows the MERGE_DATA call used to update the Players in our documentation examples database:

```
splice> CALL SYSCS_UTIL.MERGE_DATA_FROM_FILE('SPLICEBBALL', 'Players',
    'ID, Team, Name, Position, DisplayName, BirthDate',
    '/Data/DocExamplesDb/Players.csv',
    null, null, null, null, null, 0, null, true, null);
rowsImported | failedRows | files | dataSize | failedLo
g-----
---  

94          | 0        | 1     | 4720   | NONE
1 row selected
```

Example 2: Using single quotes to delimit strings

This example uses single quotes instead of double quotes as the character delimiter in the input:

```
1,This field is one line,Able  
2,'This field has two lines  
This is the second line of the field',Baker  
3,This field is also just one line,Charlie
```

Note that you must escape single quotes in SQL, which means that you actually define the character delimiter parameter with four single quotes, as follow

```
SYSCS_UTIL.MERGE_DATA_FROM_FILE('SPICE','MYTABLE',null,'data.csv','\t','','',null,null,null,0,'BAD', false, null);
```

See Importing Data Usage Examples for more examples.

See Also

- » [Our Importing Data Tutorial](#)
- » [Importing Data Usage Examples](#)
- » [SYSCS_UTIL.IMPORT_DATA](#)
- » [SYSCS_UTIL.UPSERT_DATA_FROM_FILE](#)

SYSCS_UTIL.MERGE_REGIONS

The `SYSCS_UTIL.MERGE_REGIONS` system procedure merges two adjacent Splice Machine table regions or two adjacent Splice Machine index regions.

Region names must be specified in HBase-encoded form. You can retrieve the encoded name for a region by calling the `SYSCS_UTIL.GET_ENCODED_REGION_NAME` system procedure.

You might use this procedure if you want to collect older data into a smaller set of regions to minimize the number of regions required.

Syntax

```
SYSCS_UTIL.MERGE_REGIONS( VARCHAR schemaName)
                          VARCHAR tableName,
                          VARCHAR indexName,
                          VARCHAR regionName1,
                          VARCHAR regionName2 )
```

schemaName

The name of the schema of the table.

tableName

The name of the table.

indexName

NULL or the name of the index.

Specify the name of the index if you are merging index regions; if you are merging table regions, specify NULL for this parameter.

regionName1

The **encoded** HBase name of the first of the two regions you want merged. You can call the `SYSCS_UTIL.GET_ENCODED_REGION_NAME` procedure to look up the region name for an unencoded Splice Machine table or index key.

regionName2

The **encoded** HBase name of the second of the two regions you want merged. You can call the `SYSCS_UTIL.GET_ENCODED_REGION_NAME` procedure to look up the region name for an unencoded Splice Machine table or index key.

Usage

You can merge two adjacent table regions by specifying NULL for the index name. To merge two adjacent index regions, specify both the table name and the index name.

Results

This procedure does not return a result.

If the specified regions are not adjacent, you'll see an error message, and no merging will be performed.

Examples

The following call will merge two adjacent regions of a table, after you have called `SYSCS_UTIL.GET_ENCODED_REGION_NAME` to retrieve the encoded key values for each region:

```
splice> CALL SYSCS_UTIL.MERGE_REGIONS('SPLICE','TESTTABLE', NULL,
                                         'cf0163796bba8666b1183788fc7bc31b',
                                         '4e11260fb5ae106a681574be90709449');
Statement executed.
```

And this call will merge two adjacent regions of an index, after you have called `SYSCS_UTIL.GET_ENCODED_REGION_NAME` to retrieve the encoded key values for each region::

```
splice> CALL SYSCS_UTIL.MERGE_REGIONS('SPLICE','TESTTABLE', 'SHIP_INDEX',
                                         '5a59b4a46a8a0a7180a469dbe0b40fad',
                                         '039ba9b2ecdf458b3293bd9e74e88f65');
Statement executed.
```

See Also

- » [SYSCS_UTIL.COMPACT_REGION](#)
- » [SYSCS_UTIL.GET_ENCODED_REGION_NAME](#)
- » [SYSCS_UTIL.GET_REGIONS](#)
- » [SYSCS_UTIL.GET_START_KEY](#)
- » [SYSCS_UTIL.MAJOR_COMPACT_REGION](#)

SYSCS_UTIL.SYSCS MODIFY_PASSWORD

The SYSCS_UTIL.SYSCS MODIFY_PASSWORD system procedure is called by a user to change that user's own password.

This procedure is used in conjunction with NATIVE authentication.

The derby.authentication.native.passwordLifetimeMillis property sets the password expiration time, and the derby.authentication.native.passwordLifetimeThreshold property sets the time when a user is warned that the password will expire.

Syntax

```
SYSCS_UTIL.SYSCS MODIFY_PASSWORD(IN password VARCHAR(32672))
```

password

A case-sensitive password.

Results

This procedure does not return a result.

Execute Privileges

Any user can execute this procedure.

As of this writing, your administrator must grant a user execute permission on this procedure before that user can successfully modify his or her password.

JDBC example

```
CallableStatement cs = conn.prepareCall(
    "CALL SYSCS_UTIL.SYSCS MODIFY_PASSWORD('baseball!')");
cs.execute();
cs.close();
```

SQL Example

The following example sets the current user's password to baseball!:

```
splice> CALL SYSCS_UTIL.SYSCS MODIFY_PASSWORD('baseball!');  
Statement executed
```

See Also

» [SYSCS_UTIL.SYSCS_RESET_PASSWORD](#)

SYSCS_UTIL.SYSCS_PEEK_AT_SEQUENCE Function

The SYSCS_UTIL.SYSCS_PEEK_AT_SEQUENCE function allows users to observe the instantaneous current value of a sequence generator without having to query the [SYSSEQUENCES](#) system table.

Querying the SYSSEQUENCES table does not actually return the current value; it only returns an upper bound on that value, which is the end of the chunk of sequence values that has been pre-allocated but not yet used.

The SYSCS_UTIL.SYSCS_PEEK_AT_SEQUENCE function shows you the very next value that will be returned by a `NEXT VALUE FOR` clause. Users should never directly query the SYSSEQUENCES table, because that will cause sequence generator concurrency to slow drastically.

Syntax

```
BIGINT SYSCS_UTIL.SYSCS_PEEK_AT_SEQUENCE(
    IN SchemaName VARCHAR(128),
    IN SequenceName VARCHAR(128)
)
```

SchemaName

The name of the schema.

SequenceName

The name of the sequence.

Results

Returns the next value that will be returned for the sequence.

Execute Privileges

By default, all users have execute privileges on this function.

Example

```
splice> VALUES SYSCS_UTIL.SYSCS_PEEK_AT_SEQUENCE('SPLICE', 'PlayerID_seq');
```

See Also

» [SYSSEQUENCES](#)

SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA

The `SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA` system procedure performs a major compaction on a schema. The compaction is performed on all of the tables in the schema, and on all of its index and constraint tables for each table in the schema.

Syntax

```
SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA(schemaName)
```

schemaName

A string that specifies the Splice Machine schema name to which the table belongs.

Usage

Major compaction is synchronous, which means that when you invoke this procedure from the command line, your command line prompt won't be available again until the compaction completes, which can take a little time.

Results

This procedure does not return a result.

Example

```
splice> CALL SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA('SPLICE');
Statement executed.
```

See Also

» [SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE](#)

SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE

The SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE system procedure performs a major compaction on a table. The compaction is performed on the table and on all of its index and constraint tables.

Syntax

```
SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE(  
    schemaName, tableName)
```

schemaName

A string that specifies the Splice Machine schema name to which the table belongs.

tableName

A string that specifies name of the Splice Machine table on which to perform the compaction.

Usage

Major compaction is synchronous, which means that when you invoke this procedure from the command line, your command line prompt won't be available again until the compaction completes, which can take a little time.

Results

This procedure does not return a result.

Example

```
splice> CALL SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE('SPICE','Pitching');  
Statement executed.
```

See Also

» [SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA](#)

SYSCS_UTIL.SYSCS_REFRESH_EXTERNAL_TABLE

You call the SYSCS_UTIL.SYSCS_REFRESH_EXTERNAL_TABLE system procedure to manually refresh the schema of an external table in Splice Machine that has been modified outside of Spark. When you use the external table, Spark caches its schema in memory to improve performance; as long as you are using Spark to modify the table, it is smart enough to refresh the cached schema. However, if the table schema is modified outside of Spark, you need to call SYSCS_UTIL.SYSCS_REFRESH_EXTERNAL_TABLE.

Syntax

```
SYSCS_UTIL.SYSCS_REFRESH_EXTERNAL_TABLE(
    String schemaName,
    String tableName )
```

schemaName

Specifies the schema of the table. Passing a null or non-existent schema name generates an error.

tableName

The table name.

Results

This procedure does not return a result.

Example

This refreshes the schema of the external table named myTable:

```
splice> CALL SYSCS_UTIL.SYSCS_REFRESH_EXTERNAL_TABLE('APP', 'myTable');
Statement executed.
```

See Also

- CREATE EXTERNAL TABLE

SQLJ.REMOVE_JAR

The `SQLJ.REMOVE_JAR` system procedure removes a jar file from a database.

NOTE: For more information about using JAR files, see the [Using Functions and Stored Procedures](#) section in our *Developer's Guide*.

Syntax

```
SQLJ.REMOVE_JAR(
    IN qualified_jar_name VARCHAR(32672),
    IN undeploy INTEGER
)
```

qualified_jar_name

The Splice Machine name of the jar file, qualified by the schema name. Two examples:

```
MYSHEMA.Sample1
-- a delimited identifier.
MYSHEMA."Sample2"
```

undeploy

If set to 1, this indicates the existence of an SQLJ deployment descriptor file. Splice Machine ignores this argument, so it is normally set to 0.

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

SQL Example

```
-- SQL statement
CALL SQLJ.REMOVE_JAR('SPLICE.Sample1', 0);
```

See Also

- » [SQLJ_INSTALL_JAR](#)
- » [SQLJ_REPLACE_JAR](#)

SQLJ.REPLACE_JAR

The SQLJ.REPLACE_JAR system procedure replaces a jar file in a database.

NOTE: For more information about using JAR files, see the [Using Functions and Stored Procedures](#) section in our *Developer's Guide*.

Syntax

```
SQLJ.REPLACE_JAR(
    IN jar_file_path_or-url VARCHAR(32672),
    IN qualified_jar_name VARCHAR(32672)
)
```

jar_file_path_or-url

The path or URL of the jar file to use as a replacement. A path includes both the directory and the file name (unless the file is in the current directory, in which case the directory is optional). For example:

```
d:/todays_build/tours.jar
```

qualified_jar_name

The Splice Machine name of the jar file, qualified by the schema name. Two examples:

```
MYSHEMA.Sample1
-- a delimited identifier.
MYSHEMA."Sample2"
```

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

SQL Example

```
-- SQL statement
CALL sqlj.replace_jar('c:\myjarfiles\newtours.jar', 'SPLICE.Sample1');

-- SQL statement
-- replace jar from remote location
CALL SQLJ.REPLACE_JAR('http://www.example.com/tours.jar', 'SPLICE.Sample2');
```

See Also

- » [SQLJ_INSTALL_JAR](#)
- » [SQLJ_REMOVE_JAR](#)

SYSCS_UTIL.SYSCS_RESET_PASSWORD

The SYSCS_UTIL.SYSCS_RESET_PASSWORD system procedure resets a password for a user whose password has expired or has been forgotten.

This procedure is used in conjunction with NATIVE authentication.

Syntax

```
SYSCS_UTIL.SYSCS_RESET_PASSWORD(IN userName VARCHAR(128),
                                IN password VARCHAR(32672))
```

userName

A user name that is case-sensitive if you place the name string in double quotes. This user name is an authorization identifier..

password

A case-sensitive password.

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

JDBC example

Reset the password of a user named FRED:

```
CallableStatement cs = conn.prepareCall
    ("CALL SYSCS_UTIL.SYSCS_RESET_PASSWORD(?, ?)");
cs.setString(1, "fred");
cs.setString(2, "temppassword");
cs.execute();
cs.close();
```

Reset the password of a user named FreD:

```
CallableStatement cs = conn.prepareCall
    ("CALL SYSCS_UTIL.SYSCS_RESET_PASSWORD(?, ?)");
cs.setString(1, "\\"Fred\\\"");
cs.setString(2, "temppassword");
cs.execute();
cs.close();
```

SQL Example

Reset the password of a user named FRED:

```
splice> CALL SYSCS_UTIL.SYSCS_RESET_PASSWORD('fred', 'temppassword');
Statement executed.
```

Reset the password of a user named MrBaseball:

```
splice> CALL SYSCS_UTIL.SYSCS_RESET_PASSWORD('MrBaseball', 'baseball!');
Statement executed.
```

See Also

» [SYSCS_UTIL.SYSCS_MODIFY_PASSWORD](#)

SYSCS_UTIL.SYSCS_RESTORE_DATABASE

The `SYSCS_UTIL.SYSCS_RESTORE_DATABASE` system procedure restores your database to the state it was in when a specific backup was performed, using a backup that you previously created using either the `SYSCS_UTIL_SYSCS_SCHEDULE_DAILY_BACKUP` system procedure.

You can restore your database from any previous full or incremental backup.

There are several important things to know about restoring your database from a previous backup:

- » Restoring a database **wipes out your database** and replaces it with what had been previously backed up.
- » You **cannot use your cluster** while restoring your database.
- » You **must reboot your database** after the restore is complete. See the Starting Your Database topics in this book for instructions on restarting your database.



When you restore from a backup, Splice Machine automatically determines and runs whatever sequence of restores may be required to accomplish the restoration of your database; this means that when you select an incremental backup from which to restore, Splice Machine will detect that it needs to first restore from the previous full backup and then apply any incremental restorations.

Syntax

```
SYSCS_UTIL.SYSCS_RESTORE_DATABASE( VARCHAR backupDir,
                                BIGINT backupId );
```

backupDir

Specifies the path to the directory containing the backup from which you want to restore your database. This can be a local directory if you're using the standalone version of Splice Machine, or a directory in your cluster's file system (HDFS or MapR-FS).

Relative paths are resolved based on the current user directory. To avoid confusion, we strongly recommend that you use an absolute path when specifying the backup location.

NOTE: You must specify the backup's directory when you call this procedure because, if your database has become corrupted and needs to be restored, the data in the `BACKUP .BACKUP` table (which includes the location of each backup) may also be corrupted.

backupId

The ID of the backup job from which you want to restore your database.

The system *Backing Up and Restoring* topic for more information.

Usage

Restoring your database can take a while, and has several major implications:

There are several important things to know about restoring your database from a previous backup:

- » Restoring a database **wipes out your database** and replaces it with what had been previously backed up.
- » You **cannot use your cluster** while restoring your database.
- » You **must reboot your database** after the restore is complete by first Starting Your Database.

As noted at the top of this topic: if you are restoring from an incremental backup, you must first restore from the most recent full backup, and then incrementally restore from each subsequent incremental backup. See [Example 2 below](#).

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

Examples

The following example first queries the system backup table to find the ID of the backup from which we want to restore, and then initiates the restoration.



Stop using your database before backing up, and keep in mind that restoring a database may take several minutes, depending on the size of your database.

```

splice> SELECT * FROM SYS.SYSBACKUP;
BACKUP_ID |BEGIN_TIMESTAMP          |END_TIMESTAMP           |STATUS    |FILESYSTEM
M         |SCOPE |INCR&|INCREMENTAL_PARENT_&|BACKUP_ITEM
-----
-----+
74101     |2015-11-30 17:46:41.431   |2015-11-30 17:46:56.664 |S         |./dbBackup
s/        |D      |true |40975                |30
40975     |2015-11-25 09:32:53.04   |2015-11-25 09:33:09.081 |S         |~/splicemachine
e |D      |false|-1                 |93

2 rows selected

splice> CALL SYSCS_UTIL.SYSCS_RESTORE_DATABASE('./dbBackups/' , 74101);
Statement executed.

```

Once the restoration is complete, reboot your database by the Starting Your Database.

See Also

- » [Backing Up and Restoring Databases](#)
- » [SYSCS_UTIL.SYSCS_BACKUP_DATABASE](#)
- » [SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP](#)
- » [SYSCS_UTIL.SYSCS_DELETE_BACKUP](#)
- » [SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS](#)
- » [SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP](#)
- » [SYSBACKUP](#)
- » [SYSBACKUPITEMS](#)
- » [SYSBACKUPJOBS](#)

SYSCS_UTIL.SYSCS_RESTORE_SNAPSHOT

The SYSCS_UTIL.SYSCS_RESTORE_SNAPSHOT system procedure restores a table or schema to the state it was in at the time the snapshot was created.

NOTE: Snapshots include both the data and indexes for tables.

For more information, see the [Using Snapshots](#) topic.

Syntax

```
SYSCS_UTIL.SYSCS_RESTORE_SNAPSHOT( VARCHAR(128) snapshotName );
```

snapshotName

The name of the snapshot from which you are restoring.

Results

This procedure does not return a result.

Example

The following example restores the `mySchema` schema to its state when the named snapshot was created:

```
splice> CALL SYSCS_UTIL.RESTORE_SNAPSHOT( 'snap_myschema_070417a' );
Statement executed.
```

SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP

You can use the SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP to schedule a full or incremental backup job that runs at a specified time each day.

NOTE: You specify the scheduled start hour of the backup in Greenwich Mean Time (GMT).

Note that you can subsequently cancel a scheduled backup job with the Backing Up and Restoring topic.

Syntax

```
SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_DATABASE(
    VARCHAR backupDir,
    VARCHAR(30) backupType,
    INT startHour
);
```

backupDir

Specifies the path to the directory in which you want the backup stored. This can be a local directory if you're using the standalone version of Splice Machine, or a directory in your cluster's file system (HDFS or MapR-FS).

NOTE: You must have permissions set properly to use cloud storage as a backup destination. See Backing Up to Cloud Storage in the *Administrator's Guide* for information about setting backup permissions properties.

Relative paths are resolved based on the current user directory. To avoid confusion, we strongly recommend that you use an absolute path when specifying the backup destination.

backupType

Specifies the type of backup that you want performed. This must be one of the following values: 'full' or 'incremental'; any other value produces an error and the backup is not run.

Note that if you specify 'incremental', Splice Machine checks the [SYS.SYSBACKUP](#) table to determine if there already is a backup for the system; if not, Splice Machine will perform a full backup, and subsequent backups will be incremental.

startHour

Specifies the hour (0–23) **in GMT** at which you want the backup to run each day. A value less than 0 or greater than 23 produces an error and the backup is not scheduled.

SQL Examples

The following example schedules a daily incremental backup that runs at 3 am (GMT) and gets stored in the `hdfs:///home/backup/` directory:

```
splice> CALL SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP('hdfs:///home/backup', 'incremental', 3);
Statement executed;
```

The following example schedules the same backup and stores it on AWS:

```
splice> CALL SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP('s3://backup1234', 'incremental', 3);
Statement executed.
```

And this example schedules a daily backup at 6pm (GMT) on a standalone version of Splice Machine:

```
splice> CALL SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP('./dbBackups', 'full', 18);
Statement executed.
```

See Also

- » [Backing Up and Restoring Databases in the Administrator's Guide](#)
- » [SYSCS_UTIL.SYSCS_BACKUP_DATABASE](#) built-in system procedure
- » [SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP](#) built-in system procedure
- » [SYSCS_UTIL.SYSCS_DELETE_BACKUP](#) built-in system procedure
- » [SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS](#) built-in system procedure
- » [SYSCS_UTIL.SYSCS_RESTORE_DATABASE](#) built-in system procedure
- » [SYSBACKUP](#) system table
- » [SYSBACKUPITEMS](#) system table
- » [SYSBACKUPJOBS](#) system table

SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY

Use the SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY system procedure to set or delete the value of a property of the database.

Syntax

```
SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY(
    IN key VARCHAR(128),
    IN value VARCHAR(32672)
)
```

key

The property name.

value

The new property value. If this is null, then the property with key value *key* is deleted from the database property set. If this is not null, then this *value* becomes the new value of the property. If this value is not a valid value for the property, Splice Machine uses the default value of the property.

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

JDBC example

Set the splicemachine.locks.deadlockTimeout property to a value of 10:

```
CallableStatement cs = conn.prepareCall
("CALL SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY(?, ?)");
cs.setString(1, "splicemachine.locks.deadlockTimeout");
cs.setString(2, "10");
cs.execute();
cs.close();
```

SQL Example

Set the `splicemachine.locks.deadlockTimeout` property to a value of 10:

```
splice> CALL SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY( 'splicemachine.locks.dea
dlockTimeout', '10' );
Statement executed.
```

See Also

- » [SYSCS_UTIL.SYSCS_GET_GLOBAL_DATABASE_PROPERTY](#)

SYSCS_UTIL.SYSCS_SET_LOGGER_LEVEL

The SYSCS_UTIL.SYSCS_SET_LOGGER_LEVEL system procedure changes the logging level of the specified logger.

NOTE: You can read more about Splice Machine loggers and logging levels in the [Logging](#) topic.

Syntax

```
SYSCS_UTIL.SYSCS_SET_LOGGER_LEVEL(loggerName, logLevel)
```

loggerName

A string specifying the name of the logger whose log level you want to find.

logLevel

A string specifying the new level to assign to the named logger. This must be one of the following level values, which are described in the [Logging](#) topic:

- » TRACE
- » DEBUG
- » INFO
- » WARN
- » ERROR
- » FATAL

Results

This procedure does not return a result.

Usage Notes

You can use the TRACE option of the Splice Machine StatementManager log to record the execution time of each statement:

```
splice> CALL SYSCS_UTIL.SYSCS_SET_LOGGER_LEVEL ( 'com.splicemachine.utils.SpliceUtilities', 'TRACE');
Statement executed
```

You can find all of the available loggers by using the [SYSCS_UTIL.SYSCS_GET_LOGGERS](#) system procedure.

Example

```
splice> CALL SYSCS_UTIL.SYSCS_SET_LOGGER_LEVEL( 'com.splicemachine.mrio.api', 'DEBU  
G' );  
Statement executed.
```

See Also

- » [SYSCS_UTIL.SYSCS_GET_LOGGERS](#)
- » [SYSCS_UTIL.SYSCS_SET_LOGGER_LEVEL](#)
- » [Splice Machine Logging](#)

SYSCS_UTIL.SET_PURGE_DELETED_ROWS

The SYSCS_UTIL.SET_PURGE_DELETED_ROWS system procedure enables (or disables) physical deletion of logically deleted rows from a specific table.

Syntax

```
SYSCS_UTIL.SET_PURGE_DELETED_ROWS( VARCHAR schema,
                                    VARCHAR table,
                                    VARCHAR enable );
```

schema

The name of the schema.

table

The name of the table

enable

A Boolean specifying whether or not to physically delete rows that have been logically deleted during major compaction.

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

Example

This specifies that deleted rows from `my_table` will be physically deleted when the next major compaction is run:

```
CALL SYSCS_UTIL.SET_PURGE_DELETED_ROWS('SPICE', 'my_table', true);
```

SYSCS_UTIL.SYSCS_SNAPSHOT_SCHEMA

The SYSCS_UTIL.SYSCS_SNAPSHOT_SCHEMA system procedure creates a Splice Machine snapshot of the specified schema. These snapshots can subsequently be used to restore the schema to its state at the time that a snapshot was created.

NOTE: Snapshots include both the data and indexes for tables.

For more information, see the [Using Snapshots](#) topic.

Syntax

```
SYSCS_UTIL.SYSCS_SNAPSHOT_SCHEMA( VARCHAR(128) schemaName,
                                   VARCHAR(128) snapshotName );
```

schemaName

The name of the schema for which you are creating a snapshot.

snapshotName

The name that you are assigning to this snapshot, which you can subsequently use to restore or delete the snapshot.

Results

This procedure does not return a result.



Creating a schema snapshot can require several minutes of more to complete, depending on the size of the schema.

Example

The following example creates a snapshot of the schema named mySchema:

```
splice> CALL SYSCS_UTIL.SNAPSHOT_SCHEMA('mySchema', 'snap_myschema_070417a');
Statement executed.
```

SYSCS_UTIL.SYSCS_SNAPSHOT_TABLE

The SYSCS_UTIL.SYSCS_SNAPSHOT_TABLE system procedure creates a Splice Machine snapshot of the specified table. These snapshots can subsequently be used to restore the table to its state at the time that a snapshot was created.

NOTE: Snapshots include both the data and indexes for tables.

For more information, see the [Using Snapshots](#) topic.

Syntax

```
SYSCS_UTIL.SYSCS_SNAPSHOT_TABLE( VARCHAR(128) schemaName,
                                 VARCHAR(128) tableName,
                                 VARCHAR(128) snapshotName );
```

schemaName

The name of the table's schema.

tableName

The name of the table for which you are creating a snapshot.

snapshotName

The name that you are assigning to this snapshot, which you can subsequently use to restore or delete the snapshot.

Results

This procedure does not return a result.



Creating a table snapshot can require several minutes of more to complete, depending on the size of the table.

Example

The following example creates a snapshot of the table named myTable:

```
splice> CALL SYSCS_UTIL.SNAPSHOT_SCHEMA('mySchema', 'myTable', 'snap_mySchema_070417a');
Statement executed.
```

SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX

The `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX` system procedure computes the split keys for a table or index, prior to importing that table in HFile format. You must use this procedure in conjunction with the `SYSCS_UTIL.BULK_IMPORT_HFILE` system procedure to import your data in HFile format.

Syntax

```
call SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX (
    schemaName,
    tableName,
    indexName,
    columnList | null,
    fileName,
    columnDelimiter | null,
    characterDelimiter | null,
    timestampFormat | null,
    dateFormat | null,
    timeFormat | null,
    maxBadRecords,
    badRecordDirectory | null,
    oneLineRecords | null,
    charset | null,
);
```

Parameters

The following table summarizes the parameters used by `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX` and other Splice Machine data importation procedures. Each parameter name links to a more detailed description in our Importing Data Tutorial.



The parameter values that you pass into this procedure should match the values that you use when you subsequently call the `SYSCS_UTIL.BULK_IMPORT_HFILE` procedure to perform the import.

Category	Parameter	Description	Example Value
Table Info	schemaName	The name of the schema of the table in which to import.	SPLICE
	tableName	The name of the table in which to import	playerTeams
Data Location	insertColumnList	The names, in single quotes, of the columns to import. If this is null, all columns are imported.	'ID, TEAM'

Category	Parameter	Description	Example Value
	fileOrDirectoryName	<p>Either a single file or a directory. If this is a single file, that file is imported; if this is a directory, all of the files in that directory are imported. You can import compressed or uncompressed files.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>The SYSCS_UTIL.MERGE_DATA_FROM_FILE procedure only works with single files; you cannot specify a directory name when calling SYSCS_UTIL.MERGE_DATA_FROM_FILE.</p> </div> <p>On a cluster, the files to be imported MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p>	/data/mydata/ mytable.csv 's3a://splice- benchmark-data/ flat/TPCH/100/ region'
Data Formats	oneLineRecords	A Boolean value that specifies whether (true) each record in the import file is contained in one input line, or (false) if a record can span multiple lines.	true
	charset	The character encoding of the import file. The default value is UTF-8.	null
	columnDelimiter	The character used to separate columns, Specify null if using the comma (,) character as your delimiter.	' '
	characterDelimiter	The character is used to delimit strings in the imported data.	'''
	timestampFormat	The format of timestamps stored in the file. You can set this to null if there are no time columns in the file, or if the format of any timestamps in the file match the <code>Java.sql.Timestamp</code> default format, which is: "yyyy-MM-dd HH:mm:ss".	'yyyy-MM-dd HH:mm:ss.SSZ'



All of the timestamps in the file you are importing must use the same format.

Category	Parameter	Description	Example Value
	dateFormat	The format of timestamps stored in the file. You can set this to null if there are no date columns in the file, or if the format of any dates in the file match pattern: "yyyy-MM-dd".	yyyy-MM-dd
	timeFormat	The format of time values stored in the file. You can set this to null if there are no time columns in the file, or if the format of any times in the file match pattern: "HH:mm:ss".	HH:mm:ss
Problem Logging	badRecordsAllowed	The number of rejected (bad) records that are tolerated before the import fails. If this count of rejected records is reached, the import fails, and any successful record imports are rolled back. Specify 0 to indicate that no bad records are tolerated, and specify -1 to indicate that all bad records should be logged and allowed.	25
	badRecordDirectory	<p>The directory in which bad record information is logged. Splice Machine logs information to the <import_file_name>.bad file in this directory; for example, bad records in an input file named foo.csv would be logged to a file named <i>badRecordDirectory</i>/foo.csv.bad.</p> <p>On a cluster, this directory MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p>	'importErrsDir'
Bulk HFile Import	bulkImportDirectory (outputDirectory)	<p>For SYSCS_UTIL.BULK_IMPORT_HFILE, this is the name of the directory into which the generated HFiles are written prior to being imported into your database.</p> <p>For the SYSCS_UTIL.COMPUTE_SPLIT_KEY procedure, where it is named outputDirectory, this parameter specifies the directory into which the split keys are written.</p>	hdfs:///tmp/test_hfile_import/

Category	Parameter	Description	Example Value
	skipSampling	<p>The <code>skipSampling</code> parameter is a Boolean value that specifies how you want the split keys used for the bulk HFile import to be computed. Set to <code>false</code> to have <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> automatically determine splits for you.</p> <p>This parameter is only used with the <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> system procedure.</p>	<code>false</code>

Usage

The `SYSCS_UTIL.BULK_IMPORT_HFILE` procedure needs the data that you're importing split into multiple HFiles before it actually imports the data into your database. You can achieve these splits in three ways:

- » You can call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter to `false`. `SYSCS_UTIL.BULK_IMPORT_HFILE` samples the data to determine the splits, then splits the data into multiple HFiles, and then imports the data.
- » You can split the data into HFiles with this procedure, `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX`, which both computes the keys and performs the splits. You then call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter to `true` to import your data.
- » You can split the data into HFiles by first calling the `SYSCS_UTIL.COMPUTE_SPLIT_KEY` procedure and then calling the `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS` procedure to split the table or index. You then call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter to `true` to import your data.

In all cases, `SYSCS_UTIL.BULK_IMPORT_HFILE` automatically deletes the HFiles after the import process has completed.

The Bulk HFile Import Examples section of our *Importing Data Tutorial* describes how these methods differ and provides examples of using them to import data.

Examples

The Importing Data: Bulk HFile Examples topic walks you through several examples of importing data with bulk HFiles.

See Also

- » [SYSCS_UTIL.BULK_IMPORT_HFILE](#)
- » [SYSCS_UTIL.COMPUTE_SPLIT_KEY](#)

» [SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS](#)

SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS

Before using this procedure, `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS`, you must first call the `SYSCS_UTIL.COMPUTE_SPLIT_KEY` procedure to compute the split points for the data you're importing. After computing the split keys, use this procedure to split the data into HFiles, and then call `SYSCS_UTIL.BULK_IMPORT_HFILE` system procedure to import your data in HFile format.

Syntax

```
SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS (
    schemaName,
    tableName,
    indexName,
    splitPoints
);
```

schemaName

The name of the schema of the table or index that you are splitting.

tableName

The name of the table you are splitting.

indexName

The name of the index that you are splitting. If this is null, the specified table is split; if this is non-null, the index is split instead.

splitPoints

A comma-separated list of split points for the table or index.

This is the list of split points computed by a previous call to the `SYSCS_UTIL.COMPUTE_SPLIT_KEY` procedure.

Usage

The `SYSCS_UTIL.BULK_IMPORT_HFILE` procedure needs the data that you're importing split into multiple HFiles before it actually imports the data into your database. You can achieve these splits in three ways:

- » You can call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter to `false`.
`SYSCS_UTIL.BULK_IMPORT_HFILE` samples the data to determine the splits, then splits the data into multiple HFiles, and then imports the data.
- » You can split the data into HFiles with the `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX` procedure, which both computes the keys and performs the splits. You then call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter to `true` to import your data.

- » You can split the data into HFiles by first calling the [SYSCS_UTIL.COMPUTE_SPLIT_KEY](#) procedure to compute the split points, and then call this procedure, [SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS](#) procedure to split the table or index. You then call [SYSCS_UTIL.BULK_IMPORT_HFILE](#) with the skipSampling parameter to true to import your data.

In all cases, [SYSCS_UTIL.BULK_IMPORT_HFILE](#) automatically deletes the HFiles after the import process has completed.

The Bulk HFile Import Examples section of our *Importing Data Tutorial* describes how these methods differ and provides examples of using them to import data.

See Also

- » [SYSCS_UTIL.BULK_IMPORT_HFILE](#)
- » [SYSCS_UTIL.COMPUTE_SPLIT_KEY](#)
- » [SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX](#)

SYSCS_UTIL.SYSCS_UPDATE_ALL_SYSTEM_PROCEDURES

The `SYSCS_UTIL.SYSCS_UPDATE_ALL_SYSTEM_PROCEDURES` system procedure updates the signatures of all of the system procedures in a database.

You need to call this procedure when you update to a new version of Splice Machine that includes new or updated system procedure signatures.

About System Procedures

Splice Machine uses prepared statements known as *system procedures* to access data in the system tables. Each system procedure has two parts:

- » An *implementation*, which is compiled Java byte code that is stored in the Splice jar and is included in the CLASSPATH of the Splice server.
- » A *declaration* (or *signature*), which is a CREATE PROCEDURE statement that is stored in the Splice jar file and is synchronized with the data dictionary (in the `SYSALIASES` table).

The `SYSALIASES` table is synchronized with a database when the database is first created. Thereafter, when you make changes to the system procedures, you need to call a function to keep the `SYSALIASES` table synchronized with the procedures in the Splice jar file.

If you've modified, deleted, or added a system procedure, call the `SYSCS_UTIL.SYSCS_UPDATE_SYSTEM_PROCEDURE` function, which drops the procedure from the data dictionary, and updates the dictionary with the new version in the Splice jar file.

If you've made multiple modifications to the system procedures, you can call this function, `SYSCS_UTIL.SYSCS_UPDATE_ALL_SYSTEM_PROCEDURES`, to update all of the stored declarations for a database in the data dictionary. This function drops all of the system procedures from the data dictionary and then recreates the system procedures stored in the dictionary from the definitions in the Splice jar file.

Results

This procedure does not return a result.

Syntax

```
SYSCS_UTIL.SYSCS_UPDATE_ALL_SYSTEM_PROCEDURES(schemaName)
```

schemaName

A string specifying the name of the schema that needs to be updated in the data dictionary.

Example

```
splice> call SYSCS_UTIL.SYSCS_UPDATE_ALL_SYSTEM_PROCEDURES('SYSCS_UTIL');
Statement executed.
```

See Also

- » [SYSCS_UTIL.SYSCS_UPDATE_SYSTEM_PROCEDURE](#)

SYSCS_UTIL.SYSCS_UPDATE_METADATA_STORED_STATEMENTS

The SYSCS_UPDATE_METADATA_STORED_STATEMENTS system procedure updates the execution plan for stored procedures in your database.

About System Procedures and Metadata

Splice Machine uses prepared statements known as system procedures to access data in the system tables. These procedures are cached, along with their execution plans, in the data dictionary. The cached execution plans can become sub-optimal after you issue a large number of schema-modifying DLL statements, such as defining and/or modifying a number of tables.

You typically need to call this procedure (along with the [SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE](#) procedure) whenever you update your Splice Machine software installation.

If you have called the [SYSCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS](#) system procedure to improve query speed, and performance is still sub-optimal, it is probably because the query optimizer needs some manual hints to generate an optimal execution plan.

The manual hints are stored in the metadata.properties file, which is external to the database. Versions of this file are typically supplied by Splice Machine consultants or engineers.

Use this function to update the execution plans stored in the data dictionary.

Syntax

```
SYSCS_UPDATE_METADATA_STORED_STATEMENTS()
```

Results

This procedure does not return a result.

Example

```
splice> CALL SYSCS_UPDATE_METADATA_STORED_STATEMENTS();
Statement executed.
```

See Also

» [SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE](#)

SYSCS_UTIL.SYSCS_UPDATE_SCHEMA_OWNER

The SYSCS_UTIL.SYSCS_UPDATE_SCHEMA_OWNER system procedure changes the owner of a schema.

Syntax

```
SYSCS_UTIL.SYSCS_UPDATE_SCHEMA_OWNER(  
    schemaName VARCHAR(128),  
    userName VARCHAR(128))
```

schemaName

Specifies the name of the schema..

userName

Specifies the user ID in the Splice Machine database.

Results

This procedure does not return a result.

Execute Privileges

If authentication and SQL authorization are both enabled, only the database owner has execute privileges on this function by default. The database owner can grant access to other users.

Example

```
splice> CALL SYSCS_UTIL.SYSCS_UPDATE_SCHEMA_OWNER( 'SPLICEBBALL', 'Walt');  
Statement executed.
```

SYSCS_UTIL.SYSCS_UPDATE_SYSTEM_PROCEDURE

The SYSCS_UTIL.SYSCS_UPDATE_SYSTEM_PROCEDURE system procedure updates the stored declaration of a specific system procedure in the data dictionary. Call this procedure after adding a new system procedure or modifying an existing system procedure.

About System Procedures

Splice Machine uses prepared statements known as *system procedures* to access data in the system tables. Each system procedure has two parts:

- » An *implementation*, which is compiled Java byte code that is stored in the Splice jar and is included in the CLASSPATH of the Splice server.
- » A *declaration* (or *signature*), which is a CREATE PROCEDURE statement that is stored in the Splice jar file and is synchronized with the data dictionary (in the SYSALIASES table).

The SYSALIASES table is synchronized with a database when the database is first created. Thereafter, when you make changes to the system procedures, you need to call a function to keep the SYSALIASES table synchronized with the procedures in the Splice jar file.

If you've modified, deleted, or added a system procedure, call this function, SYSCS_UTIL.SYSCS_UPDATE_SYSTEM_PROCEDURE, which drops the procedure from the data dictionary, and updates the dictionary with the new version in the Splice jar file.

Syntax

```
SYSCS_UTIL.SYSCS_UPDATE_SYSTEM_PROCEDURE(schemaName, procName)
```

schemaName

A string specifying the name of the schema that needs to be updated in the data dictionary.

procName

A string specifying the name of the system procedure whose declaration needs to be updated in the data dictionary.

Results

This procedure does not return a result.

Example

```
splice> CALL SYSCS_UTIL.SYSCS_UPDATE_SYSTEM_PROCEDURE('SYSCS_UTIL', 'IMPORT_DATA');
Statement executed.
```

See Also

- » [SYSCS_UTIL_SYSCS_UPDATE_ALL_SYSTEM_PROCEDURES](#)

SYSCS_UTIL.UPSERT_DATA_FROM_FILE

The `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` system procedure imports data to update an existing record or create a new record in your database. You can choose to import all or a subset of the columns from the input data into your database using the `insertColumnList` parameter.

After a successful import completes, a simple report displays, showing how many files were imported, and how many record imports succeeded or failed.

Selecting an Import Procedure

Splice Machine provides four system procedures for importing data:

- » The `SYSCS_UTIL.IMPORT_DATA` procedure imports each input record into a new record in your database.
- » This procedure, `SYSCS_UTIL.UPSERT_DATA_FROM_FILE`, updates existing records and adds new records to your database. It only differs from `SYSCS_UTIL.MERGE_DATA_FROM_FILE` in that upserting **overwrites** the generated or default value of a column that *is not specified* in your `insertColumnList` parameter when updating a record.
- » The `SYSCS_UTIL.MERGE_DATA_FROM_FILE` procedure updates existing records and adds new records to your database. It only differs from `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` in that merging **does not overwrite** the generated or default value of a column that *is not specified* in your `insertColumnList` parameter when updating a record.
- » The `SYSCS_BULK_IMPORT_HFILE` procedure takes advantage of HBase bulk loading to import table data into your database by temporarily converting the table file that you're importing into HFiles, importing those directly into your database, and then removing the temporary HFiles. This procedure has improved performance for large tables; however, the bulk HFile import requires extra work on your part and lacks constraint checking.

Our Importing Data Tutorial includes a decision tree and brief discussion to help you determine which procedure best meets your needs.

Syntax

```
call SYSCS_UTIL.UPSERT_DATA_FROM_FILE (
    schemaName,
    tableName,
    insertColumnList | null,
    fileOrDirectoryName,
    columnDelimiter | null,
    characterDelimiter | null,
    timestampFormat | null,
    dateFormat | null,
    timeFormat | null,
    badRecordsAllowed,
    badRecordDirectory | null,
    oneLineRecords | null,
    charset | null
);
```

Parameters

The following table summarizes the parameters used by `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` and other Splice Machine data importation procedures. Each parameter name links to a more detailed description in our Importing Data Tutorial.

Category	Parameter	Description	Example Value
Table Info	schemaName	The name of the schema of the table in which to import.	SPLICE
	tableName	The name of the table in which to import	playerTeams
Data Location	insertColumnList	The names, in single quotes, of the columns to import. If this is null, all columns are imported.	'ID, TEAM'

Category	Parameter	Description	Example Value
	<code>fileOrDirectoryName</code>	<p>Either a single file or a directory. If this is a single file, that file is imported; if this is a directory, all of the files in that directory are imported. You can import compressed or uncompressed files.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>The SYSCS_UTIL.MERGE_DATA_FROM_FILE procedure only works with single files; you cannot specify a directory name when calling SYSCS_UTIL.MERGE_DATA_FROM_FILE.</p> <p>On a cluster, the files to be imported MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p> </div>	<code>/data/mydata/mytable.csv</code> <code>'s3a://splice-benchmark-data/flat/TPCH/100/region'</code>
Data Formats	<code>oneLineRecords</code>	A Boolean value that specifies whether (<code>true</code>) each record in the import file is contained in one input line, or (<code>false</code>) if a record can span multiple lines.	<code>true</code>
	<code>charset</code>	The character encoding of the import file. The default value is UTF-8.	<code>null</code>
	<code>columnDelimiter</code>	The character used to separate columns, Specify <code>null</code> if using the comma (,) character as your delimiter.	' '
	<code>characterDelimiter</code>	The character is used to delimit strings in the imported data.	' "'
	<code>timestampFormat</code>	<p>The format of timestamps stored in the file. You can set this to <code>null</code> if there are no time columns in the file, or if the format of any timestamps in the file match the <code>Java.sql.Timestamp</code> default format, which is: "yyyy-MM-dd HH:mm:ss".</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  All of the timestamps in the file you are importing must use the same format. </div>	<code>'yyyy-MM-dd HH:mm:ss.SSZ'</code>

Category	Parameter	Description	Example Value
	dateFormat	The format of timestamps stored in the file. You can set this to null if there are no date columns in the file, or if the format of any dates in the file match pattern: "yyyy-MM-dd".	yyyy-MM-dd
	timeFormat	The format of time values stored in the file. You can set this to null if there are no time columns in the file, or if the format of any times in the file match pattern: "HH:mm:ss".	HH:mm:ss
Problem Logging	badRecordsAllowed	The number of rejected (bad) records that are tolerated before the import fails. If this count of rejected records is reached, the import fails, and any successful record imports are rolled back. Specify 0 to indicate that no bad records are tolerated, and specify -1 to indicate that all bad records should be logged and allowed.	25
	badRecordDirectory	<p>The directory in which bad record information is logged. Splice Machine logs information to the <import_file_name>.bad file in this directory; for example, bad records in an input file named foo.csv would be logged to a file named <i>badRecordDirectory</i>/foo.csv.bad.</p> <p>On a cluster, this directory MUST be on S3, HDFS (or MapR-FS). If you're using our Database Service product, files can only be imported from S3.</p>	'importErrsDir'
Bulk HFile Import	bulkImportDirectory (outputDirectory)	<p>For SYSCS_UTIL.BULK_IMPORT_HFILE, this is the name of the directory into which the generated HFiles are written prior to being imported into your database.</p> <p>For the SYSCS_UTIL.COMPUTE_SPLIT_KEY procedure, where it is named outputDirectory, this parameter specifies the directory into which the split keys are written.</p>	hdfs:///tmp/test_hfile_import/

Category	Parameter	Description	Example Value
	skipSampling	<p>The <code>skipSampling</code> parameter is a Boolean value that specifies how you want the split keys used for the bulk HFile import to be computed. Set to <code>false</code> to have <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> automatically determine splits for you.</p> <p>This parameter is only used with the <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> system procedure.</p>	<code>false</code>

Results

`SYSCS_UTIL.UPSERT_DATA_FROM_FILE` displays a summary of the import process results that looks like this:

rowsImported	failedRows	files	dataSize	failedLog
94	0	1	4720	NONE

This procedure also logs rejected record activity into `.bad` files in the `badRecordDirectory` directory; one file for each imported file.

Importing and Updating Records

What distinguishes `SYSCS_UTIL.IMPORT_DATA` from the similar `SYSCS_UTIL.UPSERT_DATA_FROM_FILE` and `SYSCS_UTIL.SYSCS_MERGED_DATA_FROM_FILE` procedures is how each works with these specific conditions:

- » You are importing only a subset of data from the input data into your table, either because the table contains less columns than does the input file, or because you've specified a subset of the columns in your `insertColumnList` parameter.
- » Inserting and updating data in a column with generated values.
- » Inserting and updating data in a column with default values.
- » Handling of missing values.

The Importing Data Tutorial: Input Handling topic describes how each of these conditions is handled by the different system procedures.

Record Import Failure Reasons

When upserting data from a file, the input file you generate must contain:

- » the columns to be changed
- » all NON_NULL columns

Typical reasons for a row (record) import to fail include:

- » Improper data expected for a column.
- » Improper number of columns of data.
- » A primary key violation: [SYSCS_UTIL.UPSERT_DATA_FROM_FILE](#) will only work correctly if the table into which you are inserting/updating has primary keys.

A few important notes:

- » Splice Machine advises you to run a full compaction (with the [SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE](#) system procedure) after importing large amounts of data into your database.
- » On a cluster, the files to be imported **MUST be on S3, HDFS (or MapR-FS)**, as must the badRecordDirectory directory. If you're using our Database Service product, files can only be imported from S3.

In addition, the files must be readable by the hbase user, and the badRecordDirectory directory must be writable by the hbase user, either by setting the user explicitly, or by opening up the permissions; for example:

```
sudo -su hdfs hadoop fs -chmod 777 /badRecordDirectory
```

Examples

This section presents a couple simple examples.

The Importing Data Usage Examples topic contains a more extensive set of examples.

Example 1: Updating our doc examples player data

This example shows the UPSERT_DATA call used to update the Players in our documentation examples database:

```
splice> CALL SYSCS_UTIL.UPSERT_DATA_FROM_FILE('SPLICEBBALL', 'Players',
  'ID, Team, Name, Position, DisplayName, BirthDate',
  '/Data/DocExamplesDb/Players.csv',
  null, null, null, null, null, 0, null, true, null);
rowsImported | failedRows | files | dataSize | failedLogs
94           | 0          | 1    | 4720   | NONE
```

1 row selected

Example 2: Importing strings with embedded special characters

This example imports a csv file that includes newline (Ctrl-M) characters in some of the input strings. We use the default double-quote as our character delimiter to import data such as the following:

```
1,This field is one line,Able
2,"This field has two lines
This is the second line of the field",Baker
3,This field is also just one line,Charlie
```

We then use the following call to import the data:

```
SYSCS_UTIL.UPSERT_DATA_FROM_FILE( 'SPICE', 'MYTABLE', null, 'data.csv',
                                '\t', null, null, null, null, 0, 'BAD', false, nul
1 );
```

We can also explicitly specify double quotes (or any other character) as our delimiter character for strings:

```
SYSCS_UTIL.UPSERT_DATA_FROM_FILE( 'SPICE', 'MYTABLE',null,'data.csv',
                                '\t', '"', null, null, null, 0,'BAD', false, nul
1);
```

See Importing Data Usage Examples for more examples.

See Also

- » [Our Importing Data Tutorial](#)
- » [Importing Data Usage Examples](#)
- » [SYSCS_UTIL.IMPORT_DATA](#)
- » [SYSCS_UTIL.MERGE_DATA_FROM_FILE](#)

SYSCS_UTIL.VACUUM

The SYSCS_UTIL.VACUUM system procedure performs the following clean-up operations:

1. Waits for all previous transactions to complete; at this point, it must be all. If it waits past a certain point, the call terminates, and you will need to run it again.
2. Gets all the conglomerates that are seen in sys.sysconglomerates (e.g. select conglomeratenumber from sys.sysconglomerates).
3. Gets a list of all of the HBase tables.
4. If an HBase table is not in the conglomerates list and is not a system table (conglomeratenumber < 1100 or 1168), then it is deleted. If this does not occur, check the splice.log.

You are ready to go when you see the Ready to accept connections message.

If you see an exception, but do not see the Ready to accept connections message, please retry the command.

Syntax

```
SYSCS_UTIL.VACUUM()
```

Example

```
splice> CALL SYSCS_UTIL.VACUUM();
Ready to accept connections.
```

System Tables

This section contains the reference documentation for the Splice Machine SQL Statements, in the following subsections:

- » Database Backups Tables
- » Database Objects Information Tables
- » Database Permissions Tables
- » Database Statistics Tables
- » System Information Tables

Since the system tables belong to the `SYS` schema, you must preface any inquiries involving these tables with the `SYS.` prefix.

NOTE: You can use the Java `java.sql.DatabaseMetaData` class to learn more about these tables.

Database Backups Tables

This is an On-Premise-Only topic! [Learn about our products](#)

These are the System Tables with backups information:

System Table	Description
SYSBACKUP	Information about each run of a backup job that has been run for the database. You can query this table to determine status information about a specific backup job.
SYSBACKUPITEMS	Information about the items backed up for each backup job.
SYSBACKUPJOBS	Information about all backup jobs that have been created for the database.

Database Objects Tables

These are the System Tables with information about database objects:

System Table	Description
SYSALIASES	Describes the procedures, functions, and user-defined types in the database.

System Table	Description
SYSCHECKS	Describes the check constraints within the current database.
SYSCOLUMNS	Describes the columns within all tables in the current database.
SYSCONSTRAINTS	Describes the information common to all types of constraints within the current database.
SYSDEPENDS	Stores the dependency relationships between persistent objects in the database.
<u>SYSFOREIGNKEYS</u>	Describes the information specific to foreign key constraints in the current database.
SYSKEYS	Describes the specific information for primary key and unique constraints within the current database.
SYSROLES	Stores the roles in the database.
SYSSCHEMAS	Describes the schemas within the current database.
SYSSEQUENCES	Describes the sequence generators in the database.
SYSSNAPSHOTS	Stores metadata for a Splice Machine snapshot.
SYSTABLES	Describes the tables and views within the current database.
SYSTRIGGERS	Describes the triggers defined for the database.
SYSVIEWS	Describes the view definitions within the current database.

Database Permissions Tables

These are the System Tables with database permissions information:

System Table	Description
SYSCOLPERMS	Stores the column permissions that have been granted but not revoked.
SYSPERMS	Describes the usage permissions for sequence generators and user-defined types.
SYSROUTINEPERMS	Stores the permissions that have been granted to routines.
SYSTABLEPERMS	Stores the table permissions that have been granted but not revoked.

Database Statistics Tables

These are the System Tables with database statistics information:

System Table	Description
SYSCOLUMNSTATISTICS	Statistics gathered for each column in each table.
SYSTABLESTATISTICS	Describes the statistics for each table within the current database.

System Information Tables

These are the System Tables with system information:

System Table	Description
SYSCONGLOMERATES	Describes the conglomerates within the current database. A conglomerate is a unit of storage and is either a table or an index.
SYSFILES	Describes jar files stored in the database.
SYSSTATEMENTS	Describes the prepared statements in the database.
SYSUSERS	Stores user credentials when NATIVE authentication is enabled.

SYSALIASES System Table

The SYSALIASES table describes the procedures, functions, and user-defined types in the database.

The following table shows the contents of the SYSALIASES system table.

SYSALIASES system table

Column Name	Type	Length	Nullable	Contents
ALIASID	CHAR	36	NO	Unique identifier for the alias
ALIAS	VARCHAR	128	NO	Alias (in the case of a user-defined type, the name of the user-defined type)
SCHEMAID	CHAR	36	YES	Reserved for future use
JAVACLASSNAME	LONG VARCHAR	2,147,483,647	NO	The Java class name
ALIASTYPE	CHAR	1	NO	'F' (function), 'P' (procedure), 'A' (user-defined type)
NAMESPACE	CHAR	1	NO	'F' (function), 'P' (procedure), 'A' (user-defined type)
SYSTEMALIAS	BOOLEAN	1	NO	YES (system supplied or built-in alias) NO (alias created by a user)
ALIASINFO	<i>org.apache.Splice Machine.catalog.AliasInfo</i> This class is not part of the public API.	-1	YES	A Java interface that encapsulates the additional information that is specific to an alias
SPECIFICNAME	VARCHAR	128	NO	System-generated identifier

See Also

- » [About System Tables](#)

SYSBACKUP System Table

The SYSBACKUP table maintains information about each database backup. You can query this table to find the ID of and details about a backup that was run at a specific time.

SYSBACKUP system table

Column Name	Type	Length	Nullable	Contents
BACKUP_ID	BIGINT	19	NO	The backup ID
BEGIN_TIMESTAMP	TIMESTAMP	29	NO	The start time of the backup
END_TIMESTAMP	TIMESTAMP	29	YES	The end time of the backup
STATUS	VARCHAR	10	NO	The status of the backup
FILESYSTEM	VARCHAR	32642	NO	The backup destination directory
SCOPE	VARCHAR	10	NO	The scope of the backup: database, schemas, tables, etc. The current allowable values are: <ul style="list-style-type: none"> » D for the entire database
INCREMENTAL_BACKUP	BOOLEAN	1	NO	YES for incremental backups, NO for full backups NOTE: Incremental backups are not yet available.
INCREMENTAL_PARENT_BACKUP_ID	BIGINT	19	YES	For an incremental backup, this is the BACKUP_ID of the previous backup on which this incremental backup is based. For full backups, this is –1. NOTE: Incremental backups are not yet available.
BACKUP_ITEM	INTEGER	10	YES	The number of tables that were backed up.

SYSBACKUPITEMS System Table

The SYSBACKUPITEMS table maintains information about each item (table) backed up during a backup.

SYSBACKUPITEMS system table

Column Name	Type	Length	Nullable	Contents
BACKUP_ID	BIGINT	19	NO	The backup ID.
ITEM	VARCHAR	32642	NO	The name of the item.
BEGIN_TIMESTAMP	TIMESTAMP	29	NO	The start time of backing up this item.
END_TIMESTAMP	TIMESTAMP	29	YES	The end time of backing up this item.
SNAPSHOT_NAME	VARCHAR	32642	NO	The name of the snapshot associated with this item.

SYSBACKUPJOBS System Table

The SYSBACKUPJOBS table maintains information about all backup jobs that have been created for the database.

SYSBACKUPJOBS system table

Column Name	Type	Length	Nullable	Contents
JOB_ID	BIGINT	19	NO	The ID of this backup job.
FILESYSTEM	VARCHAR	4000	NO	The backup destination directory.
TYPE	VARCHAR	32	NO	The backup type; possible values are: <code>incremental</code> or <code>full</code> .
HOUR_OF_DAY	INTEGER	10	NO	The regularly scheduled start time (in GMT hours) of the backup job if it is a daily backup.
BEGIN_TIMESTAMP	TIMESTAMP	29	NO	When this job was submitted.

SYSCHECKS System Table

The SYSCHECKS table describes the check constraints within the current database.

The following table shows the contents of the SYSCHECKS system table.

SYSCHECKS system table

Column Name	Type	Length	Nullable	Contents
CONSTRAINTID	CHAR	36	NO	Unique identifier for the constraint
CHECKDEFINITION	LONG VARCHAR	32,700	NO	Text of check constraint definition
REFERENCEDCOLUMNS	<i>com.splicemachine.db.catalog.ReferencedColumns</i> This class is not part of the public API.	-1	NO	Description of the columns referenced by the check constraint

See Also

- » [About System Tables](#)

SYSCOLPERMS System Table

The SYSCOLPERMS table stores the column permissions that have been granted but not revoked.

All of the permissions for one (GRANTEE, TABLEID, TYPE, GRANTOR) combination are specified in a single row in the SYSCOLPERMS table. The keys for the SYSCOLPERMS table are:

- » Primary key (GRANTEE, TABLEID, TYPE, GRANTOR)
- » Unique key (COLPERMSID)
- » Foreign key (TABLEID references SYS.SYSTABLES)

The following table shows the contents of the SYSCOLPERMS system table.

SYSCOLPERMS system table

Column Name	Type	Length	Nullable	Contents
COLPERMSID	CHAR	36	NO	Used by the dependency manager to track the dependency of a view, trigger, or constraint on the column level permissions
GRANTEE	VARCHAR	128	NO	The authorization ID of the user or role to which the privilege was granted
GRANTOR	VARCHAR	128	NO	The authorization ID of the user who granted the privilege. Privileges can be granted only by the object owner
TABLEID	CHAR	36	NO	The unique identifier for the table on which the permissions have been granted
TYPE	CHAR	1	NO	<p>If the privilege is non-grantable, the valid values are:</p> <ul style="list-style-type: none"> » 's' for SELECT » 'u' for UPDATE » 'r' for REFERENCES <p>If the privilege is grantable, the valid values are:</p> <ul style="list-style-type: none"> » 'S' for SELECT » 'U' for UPDATE » 'R' for REFERENCES

Column Name	Type	Length	Nullable	Contents
COLUMNS	<i>org.apache.SpliceMachine.iapi.services.io.FormatableBitSet</i> This class is not part of the public API.	-1	NO	A list of columns to which the privilege applies

See Also

- » [About System Tables](#)

SYSCOLUMNS System Table

The SYSCOLUMNS table describes the columns within all tables in the current database.

The following table shows the contents of the SYSCOLUMNS system table.

SYSCOLUMNS system table

Column Name	Type	Length	Nullable	Contents
REFERENCEID	CHAR	36	NO	Identifier for table (join with SYSTABLES . TABLEID)
COLUMNNAME	VARCHAR	128	NO	Column or parameter name
COLUMNNUMBER	INTEGER	10	NO	The position of the column within the table
COLUMNDATATYPE	<i>com.splicemachine.db.catalog.TypeDescriptor</i> This class is not part of the public API.	-1	NO	System type that describes precision, length, scale, nullability, type name, and storage type of data. For a user-defined type, this column can hold a <i>TypeDescriptor</i> that refers to the appropriate type alias in SYS . SYSALIASES.
COLUMNDEFAULT	<i>java.io.Serializable</i>	-1	YES	For tables, describes default value of the column. The <i>toString()</i> method on the object stored in the table returns the text of the default value as specified in the CREATE TABLE or ALTER TABLE statement.
COLUMNDEFAULTID	CHAR	36	YES	Unique identifier for the default value

Column Name	Type	Length	Nullable	Contents
AUTOINCREMENTVALUE	BIGINT	19	YES	What the next value for column will be, if the column is an identity column
AUTOINCREMENTSTART	BIGINT	19	YES	Initial value of column (if specified), if it is an identity column
AUTOINCREMENTINC	BIGINT	19	YES	Amount column value is automatically incremented (if specified), if the column is an identity column
COLLECTSTATS	BOOLEAN	1	YES	Whether or not to collect statistics on the table.

See Also

» [About System Tables](#)

SYSCOLUMNSTATISTICS System Table

The SYSCOLUMNSTATISTICS table view describes the statistics for a specific table column within the current database.

NOTE: SYS.SYSCOLUMNSTATISTICS is a system view.

The following table shows the contents of the SYSCOLUMNSTATISTICS system table.

SYSCOLUMNSTATISTICS system table

Column Name	Type	Length	Nullable	Contents
SCHEMANAME	VARCHAR	32672	YES	The name of the schema.
TABLENAME	VARCHAR	32672	YES	The name of the table.
COLUMNNAME	VARCHAR	32672	YES	The name of the column.
CARDINALITY	BIGINT	19	YES	The estimated number of distinct values for the column.
NULL_COUNT	BIGINT	19	YES	The number of rows in the table that have NULL for the column.
NULL_FRACTION	REAL	23	YES	The ratio of NULL records to all records: $\text{NULL_COUNT} / \text{TOTAL_ROW_COUNT}$
MIN_VALUE	VARCHAR	32672	YES	The minimum value for the column.
MAX_VALUE	VARCHAR	32672	YES	The maximum value for the column.
QUANTILES	VARCHAR	32672	YES	The quantiles statistics sketch for the column.
FREQUENCIES	VARCHAR	32672	YES	The frequencies statistics sketch for the column.
THETA	VARCHAR	32672	YES	The theta statistics sketch for the column.

The QUANTILES, FREQUENCIES, and THETA values are all sketches computed using the Yahoo Data Sketches library, which you can read about here: <https://datasketches.github.io/>

See Also

- » [About System Tables](#)
- » [SYSTABLESTATISTICS](#)

SYSCONGLOMERATES System Table

The SYSCONGLOMERATES table describes the conglomerates within the current database. A conglomerate is a unit of storage and is either a table or an index.

The following table shows the contents of the SYSCONGLOMERATES system table.

SYSCONGLOMERATES system table

Column Name	Type	Length	Nullable	Contents
SCHEMAID	CHAR	36	NO	Schema ID for the conglomerate
TABLEID	CHAR	36	NO	Identifier for table (join with SYSTABLES.TABLEID)
CONGLOMERATENUMBER	BIGINT	19	NO	Conglomerate ID for the conglomerate (heap or index)
CONGLOMERATENAME	VARCHAR	128	YES	Index name, if conglomerate is an index, otherwise the table ID
ISINDEX	BOOLEAN	1	NO	Whether or not conglomerate is an index
DESCRIPTOR	<i>org.apache.splicemachine.catalog.IndexDescriptor</i> This class is not part of the public API.	-1	YES	System type describing the index
ISCONSTRAINT	BOOLEAN	1	YES	Whether or not the conglomerate is a system-generated index enforcing a constraint
CONGLOMERATEID	CHAR	36	NO	Unique identifier for the conglomerate

See Also

- » About System Tables

SYSCONSTRAINTS System Table

The SYSCONSTRAINTS table describes the information common to all types of constraints within the current database (currently, this includes primary key, unique, and check constraints).

The following table shows the contents of the SYSCONSTRAINTS system table.

SYSCONSTRAINTS system table

Column Name	Type	Length	Nullable	Contents
CONSTRAINTID	CHAR	36	NO	Unique identifier for constraint
TABLEID	CHAR	36	NO	Identifier for table (join with SYSTABLES . TABLEID)
CONSTRAINTNAME	VARCHAR	128	NO	Constraint name (internally generated if not specified by user)
TYPE	CHAR	1	NO	Possible values: » 'P' for primary key) » 'U' for unique) » 'C' for check)
SCHEMAID	CHAR	36	NO	Identifier for schema that the constraint belongs to (join with SYSSCHEMAS . SCHEMAID)
STATE	CHAR	1	NO	Possible values: » 'E' for enabled » 'D' for disabled
REFERENCECOUNT	INTEGER	10	NO	The count of the number of foreign key constraints that reference this constraint; this number can be greater than zero only or PRIMARY KEY and UNIQUE constraints

See Also

- » About System Tables

SYSDEPENDS System Table

The SYSDEPENDS table stores the dependency relationships between persistent objects in the database.

Persistent objects can be dependents or providers. Dependents are objects that depend on other objects. Providers are objects that other objects depend on.

- » Dependents are views, constraints, or triggers.
- » Providers are tables, conglomerates, constraints, or privileges.

The following table shows the contents of the SYSDEPENDS system table.

SYSDEPENDS system table

Column Name	Type	Length	Nullable	Contents
DEPENDENTID	CHAR	36	NO	A unique identifier for the dependent
DEPENDENTFINDER	<i>com.splicemachine.db.catalog.TypeDescriptor</i> This class is not part of the public API.	-1	NO	A system type that describes the view, constraint, or trigger that is the dependent
PROVIDERID	CHAR	36	NO	A unique identifier for the provider
PROVIDERFINDER	<i>com.splicemachine.db.catalog.TypeDescriptor</i> This class is not part of the public API.	-1	NO	A system type that describes the table, conglomerate, constraint, and privilege that is the provider

SYSFILES System Table

The SYSFILES table describes jar files stored in the database.

The following table shows the contents of the SYSFILES system table.

SYSFILES system table

Column Name	Type	Length	Nullable	Contents
FILEID	CHAR	36	NO	Unique identifier for the jar file
SCHEMAID	CHAR	36	NO	ID of the jar file's schema (join with SYSSCHEMAS . SCHEMAID)
FILENAME	VARCHAR	128	NO	SQL name of the jar file
GENERATIONID	BIGINT	19	NO	Generation number for the file. When jar files are replaced, their generation identifiers are changed.

See Also

- » [About System Tables](#)

SYSFOREIGNKEYS System Table

The SYSFOREIGNKEYS table describes the information specific to foreign key constraints in the current database.

Splice Machine generates a backing index for each foreign key constraint. The name of this index is the same as SYSFOREIGNKEYS.CONGLOMERATEID.

The following table shows the contents of the SYSFOREIGNKEYS system table.

SYSFOREIGNKEYS system table

Column Name	Type	Length	Nullable	Contents
CONSTRAINTID	CHAR	36	NO	Unique identifier for the foreign key constraint (join with SYSCONSTRAINTS.CONSTRAINTID)
CONGLOMERATEID	CHAR	36	NO	Unique identifier for index backing up the foreign key constraint (join with SYSCONGLOMERATES.CONGLOMERATEID)
KEYCONSTRAINTID	CHAR	36	NO	Unique identifier for the primary key or unique constraint referenced by this foreign key SYSKEYS.CONSTRAINTID or SYSCONSTRAINTS.CONSTRAINTID)
DELETERULE	CHAR	1	NO	Possible values: 'R' for NO ACTION (default) 'S' for RESTRICT 'C' for CASCADE 'U' for SET NULL
UPDATERULE	CHAR	1	NO	Possible values: 'R' for NO ACTION (default) 'S' for RESTRICT

SYSKEYS System Table

The SYSKEYS table describes the specific information for primary key and unique constraints within the current database.

Splice Machine generates an index on the table to back up each such constraint. The index name is the same as SYSKEYS.CONGLOMERATEID.

The following table shows the contents of the SYSKEYS system table.

SYSKEYS system table

Column Name	Type	Length	Nullable	Contents
CONSTRAINTID	CHAR	36	NO	Unique identifier for constraint
CONGLOMERATEID	CHAR	36	NO	Unique identifier for backing index

See Also

- » [About System Tables](#)

SYSPERMS System Table

The SYSPERMS table describes the USAGE permissions for sequence generators and user-defined types.

The following table shows the contents of the SYSPERMS system table.

SYSPERMS system table

Column Name	Type	Length	Nullable	Contents
UUID	CHAR	36	NO	The unique ID of the permission. This is the primary key.
OBJECTTYPE	VARCHAR	36	NO	<p>The kind of object receiving the permission. The only valid values are:</p> <ul style="list-style-type: none"> » 'SEQUENCE' » 'TYPE'
OBJECTID	CHAR	36	NO	<p>The UUID of the object receiving the permission.</p> <p>For sequence generators, the only valid values are SEQUENCEIDs in the SYS.SYSSEQUENCES table.</p> <p>For user-defined types, the only valid values are ALIASIDs in the SYS.SYSALIASES table if the SYSALIASES row describes a user-defined type.</p>
PERMISSION	CHAR	36	NO	The type of the permission. The only valid value is 'USAGE'.
GRANTOR	VARCHAR	128	NO	The authorization ID of the user who granted the privilege. Privileges can be granted only by the object owner.
GRANTEE	VARCHAR	128	NO	The authorization ID of the user or role to which the privilege was granted
ISGRANTABLE	CHAR	1	NO	<p>If the GRANTEE is the owner of the sequence generator or user-defined type, this value is 'Y'.</p> <p>If the GRANTEE is not the owner of the sequence generator or user-defined type, this value is 'N'.</p>

SYSROLES System Table

The SYSROLES table stores the roles in the database.

A row in the SYSROLES table represents one of the following:

- » A role definition (the result of a `CREATE ROLE` statement)
- » A role grant

The keys for the SYSROLES table are:

- » Primary key (GRANTEE, ROLEID, GRANTOR)
- » Unique key (UUID)

The following table shows the contents of the SYSROLES system table.

SYSROLES system table

Column Name	Type	Length	Nullable	Contents
UUID	CHAR	36	NO	A unique identifier for this role
ROLEID	VARCHAR	128	NO	The role name, after conversion to case normal form
GRANTEE	VARCHAR	128	NO	If the row represents a role grant, this is the authorization identifier of a user or role to which this role is granted. If the row represents a role definition, this is the database owner's user name.
GRANTOR	VARCHAR	128	NO	This is the authorization identifier of the user that granted this role. If the row represents a role definition, this is the authorization identifier <code>_SYSTEM</code> . If the row represents a role grant, this is the database owner's user name (since only the database owner can create and grant roles).
WITHADMINOPTION	CHAR	1	NO	A role definition is modelled as a grant from <code>_SYSTEM</code> to the database owner, so if the row represents a role definition, the value is always ' <code>Y</code> '. This means that the creator (the database owner) is always allowed to grant the newly created role. Currently roles cannot be granted <code>WITH ADMIN OPTION</code> , so if the row represents a role grant, the value is always ' <code>N</code> '.
ISDEF	CHAR	1	NO	If the row represents a role definition, this value is ' <code>Y</code> '. If the row represents a role grant, the value is ' <code>N</code> '.

See Also

- » [About System Tables](#)
- » [CURRENT_ROLE function](#)
- » [CREATE_ROLE statement](#)
- » [DROP_ROLE statement](#)
- » [GRANT statement](#)
- » [REVOKE statement](#)
- » [SET ROLE statement](#)

SYSROUTINEPERMS System Table

The SYSROUTINEPERMS table stores the permissions that have been granted to routines.

Each routine EXECUTE permission is specified in a row in the SYSROUTINEPERMS table. The keys for the SYSROUTINEPERMS table are:

- » Primary key (GRANTEE, ALIASID, GRANTOR)
- » Unique key (ROUTINEPERMSID)
- » Foreign key (ALIASID references SYS.SYSALIASES)

The following table shows the contents of the SYSROUTINEPERMS system table.

SYSROUTINEPERMS system table

Column Name	Type	Length	Nullable	Contents
ROUTINEPERMSID	CHAR	36	NO	Used by the dependency manager to track the dependency of a view, trigger, or constraint on the routine level permissions
GRANTEE	VARCHAR	128	NO	The authorization ID of the user or role to which the privilege is granted
GRANTOR	VARCHAR	128	NO	The authorization ID of the user who granted the privilege. Privileges can be granted only by the object owner.
ALIASID	CHAR	36	NO	<p>The ID of the object of the required permission.</p> <p>If PERMTYPE= 'E' , the ALIASID is a reference to the SYS.SYSALIASES table.</p> <p>Otherwise, the ALIASID is a reference to the SYS.SYSTABLES table.</p>
GRANTOPTION	CHAR	1	NO	Specifies if the GRANTEE is the owner of the routine. Valid values are 'Y' and 'N'.

SYSSCHEMAS System Table

The SYSSCHEMAS table describes the schemas within the current database.

The following table shows the contents of the SYSSCHEMAS system table.

SYSSCHEMAS system table

Column Name	Type	Length	Nullable	Contents
SCHEMAID	CHAR	36	NO	Unique identifier for the schema
SCHEMANAME	VARCHAR	128	NO	Schema name
AUTHORIZATIONID	VARCHAR	128	NO	The authorization identifier of the owner of the schema

See Also

- » [About System Tables](#)

SYSSEQUENCES System Table

The SYSSEQUENCES table describes the sequence generators in the database.

NOTE: Users should not directly query the SYSSEQUENCES table, because that will slow down the performance of sequence generators. Instead, users should call the `SYSICS_UTIL.SYSCS_PEEK_AT_SEQUENCE` system function

The following table shows the contents of the SYSSEQUENCES system table.

SYSSEQUENCES system table

Column Name	Type	Length	Nullable	Contents
SEQUENCEID	CHAR	36	NO	The ID of the sequence generator. This is the primary key.
SEQUENCENAME	VARCHAR	128	NO	The name of the sequence generator. There is a unique index on the SEQUENCENAME column.
SCHEMAID	CHAR	36	NO	The ID of the schema that contains the sequence generator. There is a column to SYSSCHMID.
SEQUENCEDATATYPE	<code>com.splicemachine.db.catalog.TypeDescriptor</code>	-1	NO	System type that defines the length, scale, nullable, and storage type of the sequence value.
CURRENTVALUE	BIGINT	19	YES	The current value of the sequence generator. This is not the actual value obtained by calling <code>SYSICS_UTIL.SYSCS_PEEK_AT_SEQUENCE</code> . It is the end of the range of values preallocated in order to provide an initial value of this sequence.
STARTVALUE	BIGINT	19	NO	This column is NULL if the sequence generator is exhausted and no more numbers.

Column Name	Type	Length	Nullable	Contents
MINIMUMVALUE	BIGINT	19	NO	The minimum value of the sequence.
MAXIMUMVALUE	BIGINT	19	NO	The maximum value of the sequence.
INCREMENT	BIGINT	19	NO	The step size of the sequence.
CYCLEOPTION	CHAR	1	NO	If the sequence generates a cycle, the value is 'Y'. If the sequence generates a linear sequence, the value is 'N'.

See Also

» [About System Tables](#)

SYSSNAPSHOTS System Table

The SYSSNAPSHOTS table describes the metadata for system snapshots.

The following table shows the contents of the SYSSNAPSHOTS system table.

NOTE: Table snapshots both the data and indexes for the table.

SYSSNAPSHOTS system table

Column Name	Type	Length	Nullable	Contents
SNAPSHOTNAME	VARCHAR	128	NO	The name of the snapshot
SCHEMANAME	VARCHAR	128	NO	Schema name
OBJECTNAME	VARCHAR	128	NO	The name of the table or index
CONGLOMERATENUMBER	BIGINT	19	NO	The conglomerate number of the object
CREATIONTIME	TIMESTAMP	29	NO	The time at which the snapshot was taken
LASTRESTORETIME	TIMESTAMP	29	NO	The time at which the snapshot was most recently restored

See Also

- » [About System Tables](#)

SYSSTATEMENTS System Table

The SYSSTATEMENTS table describes the prepared statements in the database.

The table contains one row per stored prepared statement.

The following table shows the contents of the SYSSTATEMENTS system table.

SYSSTATEMENTS system table

Column Name	Type	Length	Nullable	Contents
STMTID	CHAR	36	NO	Unique identifier for the statement
STMTNAME	VARCHAR	128	NO	Name of the statement
SCHEMAID	CHAR	36	NO	The schema in which the statement resides
TYPE	CHAR	1	NO	Always 'S'
VALID	BOOLEAN	1	NO	Whether or not the statement is valid
TEXT	LONG VARCHAR	32,700	NO	Text of the statement
LASTCOMPILED	TIMESTAMP	29	YES	Time that the statement was compiled
COMPILATIONSCHHEMAID	CHAR	36	NO	ID of the schema containing the statement
USINGTEXT	LONG VARCHAR	32,700	YES	Text of the USING clause of the CREATE STATEMENT and ALTER STATEMENT statements

See Also

- » [About System Tables](#)

SYSTABLEPERMS System Table

The SYSTABLEPERMS table stores the table permissions that have been granted but not revoked.

All of the permissions for one (GRANTEE, TABLEID, GRANTOR) combination are specified in a single row in the SYSTABLEPERMS table. The keys for the SYSTABLEPERMS table are:

- » Primary key (GRANTEE, TABLEID, GRANTOR)
- » Unique key (TABLEPERMSID)
- » Foreign key (TABLEID references SYS.SYSTABLES)

The following table shows the contents of the SYSTABLEPERMS system table.

SYSTABLEPERMS system table

Column Name	Type	Length	Nullable	Contents
TABLEPERMSID	CHAR	36	NO	Used by the dependency manager to track the dependency of a view, trigger, or constraint on the table level permissions
GRANTEE	VARCHAR	128	NO	The authorization ID of the user or role to which the privilege is granted
GRANTOR	VARCHAR	128	NO	The authorization ID of the user who granted the privilege. Privileges can be granted only by the object owner
TABLEID	CHAR	36	NO	The unique identifier for the table on which the permissions have been granted
SELECTPRIV	CHAR	1	NO	Specifies if the SELECT permission is granted. The valid values are: <ul style="list-style-type: none"> » 'y' (non-grantable privilege) » 'Y' (grantable privilege) » 'N' (no privilege)
DELETEPRIV	CHAR	1	NO	Specifies if the DELETE permission is granted. The valid values are: <ul style="list-style-type: none"> » 'y' (non-grantable privilege) » 'Y' (grantable privilege) » 'N' (no privilege)

Column Name	Type	Length	Nullable	Contents
INSERTPRIV	CHAR	1	NO	<p>Specifies if the INSERT permission is granted. The valid values are:</p> <ul style="list-style-type: none"> » 'y' (non-grantable privilege) » 'Y' (grantable privilege) » 'N' (no privilege)
UPDATEPRIV	CHAR	1	NO	<p>Specifies if the UPDATE permission is granted. The valid values are:</p> <ul style="list-style-type: none"> » 'y' (non-grantable privilege) » 'Y' (grantable privilege) » 'N' (no privilege)
REFERENCESPRIV	CHAR	1	NO	<p>Specifies if the REFERENCE permission is granted. The valid values are:</p> <ul style="list-style-type: none"> » 'y' (non-grantable privilege) » 'Y' (grantable privilege) » 'N' (no privilege)
TRIGGERPRIV	CHAR	1	NO	<p>Specifies if the TRIGGER permission is granted. The valid values are:</p> <ul style="list-style-type: none"> » 'y' (non-grantable privilege) » 'Y' (grantable privilege) » 'N' (no privilege)

See Also

- » [About System Tables](#)

SYSTABLES System Table

The SYSTABLES table describes the tables and views within the current database.

The following table shows the contents of the SYSTABLES system table.

SYSTABLES system table

Column Name	Type	Length	Nullable	Contents
TABLEID	CHAR	36	NO	Unique identifier for table or view
TABLENAME	VARCHAR	128	NO	Table or view name
TABLETYPE	CHAR	1	NO	Possible values are: » 'S' (system table) » 'T' (user table) » 'A' (synonym) » 'V' (view)
SCHEMAID	CHAR	36	NO	Schema ID for the table or view
LOCKGRANULARITY	CHAR	1	NO	Lock granularity for the table: » 'T' (table level locking) » 'R' (row level locking, the default)
VERSION	VARCHAR	128	YES	Version ID.

See Also

- » About System Tables

SYSTABLESTATISTICS System Table

The SYSTABLESTATISTICS table view describes the statistics for tables within the current database.

NOTE: SYS.SYSTABLESTATISTICS is a system view.

The following table shows the contents of the SYSTABLESTATISTICS system table.

SYSTABLESTATISTICS system table

Column Name	Type	Length	Nullable	Contents
SCHEMANAME	VARCHAR	32672	YES	The name of the schema
TABLENAME	VARCHAR	32672	YES	The name of the table
CONGLOMERATENAME	VARCHAR	32672	YES	The name of the table
TOTAL_ROW_COUNT	BIGINT	19	YES	The total number of rows in the table
AVG_ROW_COUNT	BIGINT	19	YES	The average number of rows in the table
TOTAL_SIZE	BIGINT	19	YES	The total size of the table
NUM_PARTITIONS	BIGINT	19	YES	The number of partitions ¹ for which statistics were collected.
AVG_PARTITION_SIZE	BIGINT	19	YES	The average size of a single partition ¹ , in bytes.
ROW_WIDTH	BIGINT	19	YES	<p>The <i>maximum average</i> of the widths of rows in the table, across all partitions, in bytes.</p> <p>Each partition records the average width of a single row. This value is the maximum of those averages across all partitions.</p>

Column Name	Type	Length	Nullable	Contents								
STATS_TYPE	INTEGER	10	YES	<p>The type of statistics, which is one of these values:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>0</td><td>Full table (not sampled) statistics that reflect the unmerged partition values.</td></tr> <tr> <td>1</td><td>Sampled statistics that reflect the unmerged partition values.</td></tr> <tr> <td>2</td><td>Full table (not sampled) statistics that reflect the table values after all partitions have been merged.</td></tr> <tr> <td>3</td><td>Sampled statistics that reflect the table values after all partitions have been merged.</td></tr> </table> <p>If this value is <code>NULL</code>, 0 is used.</p>	0	Full table (not sampled) statistics that reflect the unmerged partition values.	1	Sampled statistics that reflect the unmerged partition values.	2	Full table (not sampled) statistics that reflect the table values after all partitions have been merged.	3	Sampled statistics that reflect the table values after all partitions have been merged.
0	Full table (not sampled) statistics that reflect the unmerged partition values.											
1	Sampled statistics that reflect the unmerged partition values.											
2	Full table (not sampled) statistics that reflect the table values after all partitions have been merged.											
3	Sampled statistics that reflect the table values after all partitions have been merged.											
SAMPLE_FRACTION	DOUBLE	52	YES	<p>The sampling percentage, expressed as 0.0 to 1.0,</p> <ul style="list-style-type: none"> » If <code>statsType=0</code> (full statistics), this value is not used, and is shown as 0. » If <code>statsType=1</code>, this value is the percentage or rows to be sampled. A value of 0 means no rows, and a value of 1 means all rows (full statistics). 								

¹Currently, a *partition* is equivalent to a region. In the future, we may use a more finely-grained definition for partition.

See Also

- » [About System Tables](#)
- » [SYSCOLUMNSTATISTICS system table](#)

SYSTRIGGERS System Table

The SYSTRIGGERS table describes the database's triggers.

The following table shows the contents of the SYSTRIGGERS system table.

SYSTRIGGERS system table

Column Name	Type	Length	Nullable	Contents
TRIGGERID	CHAR	36	NO	Unique identifier for the trigger
TRIGGERNAME	VARCHAR	128	NO	Name of the trigger
SCHEMAID	CHAR	36	NO	ID of the trigger's schema (join with SYSSCHEMAS . SCHEMAID)
CREATIONTIMESTAMP	TIMESTAMP	29	NO	Time the trigger was created
EVENT	CHAR	1	NO	Possible values are: » 'U' for update » 'D' for delete » 'I' for insert
FIRINGTIME	CHAR	1	NO	Possible values are: » 'B' for before » 'A' for after
TYPE	CHAR	1	NO	Possible values are: » 'R' for row » 'S' for statement
STATE	CHAR	1	NO	Possible values are: » 'E' for enabled » 'D' for disabled

Column Name	Type	Length	Nullable	Contents
TABLEID	CHAR	36	NO	ID of the table on which the trigger is defined
WHENSTMTID	CHAR	36	YES	Used only if there is a WHEN clause (not yet supported)
ACTIONSTMTID	CHAR	36	YES	ID of the stored prepared statement for the triggered-SQL-statement (join with SYSSTATEMENTS . STMTID)
REFERENCEDCOLUMNS	<i>org.apache.Splice Machine.catalog.ReferencedColumns</i> This class is not part of the public API.	-1	YES	Descriptor of the columns to be updated, if this trigger is an update trigger (that is, if the EVENT column contains 'U')
TRIGGERDEFINITION	LONG VARCHAR	2,147,483,647	YES	Text of the action SQL statement
REFERENCINGOLD	BOOLEAN	1	YES	Whether or not the OLDREFERENCINGNAME, if non-null, refers to the OLD row or table
REFERENCINGNEW	BOOLEAN	1	YES	Whether or not the NEWREFERENCINGNAME, if non-null, refers to the NEW row or table
OLDREFERENCINGNAME	VARCHAR	128	YES	Pseudoname as set using the REFERENCING OLD AS clause
NEWREFERENCINGNAME	VARCHAR	128	YES	Pseudoname as set using the REFERENCING NEW AS clause

Any SQL text that is part of a triggered-SQL-statement is compiled and stored in the SYSSTATEMENTS table. ACTIONSTMTID and WHENSTMTID are foreign keys that reference SYSSTATEMENTS . STMTID. The statements for a trigger are always in the same schema as the trigger.

SYSUSERS System Table

The SYSUSERS table stores user credentials when NATIVE authentication is enabled.

When SQL authorization is enabled (as it is, for instance, when NATIVE authentication is on) only the database owner can SELECT from this table, and no one, not even the database owner, can SELECT the PASSWORD column.

The following table shows the contents of the SYSUSERS system table.

SYSUSERS system table

Column Name	Type	Length	Nullable	Contents
USERNAME	VARCHAR	128	NO	The user's name, the value of the user attribute on a connection URL.
HASHINGScheme	VARCHAR	32672	NO	Describes how the password is hashed.
PASSWORD	VARCHAR	32672	NO	The password after applying the HASHINGScheme.
LASTMODIFIED	TIMESTAMP	29	NO	The time when the password was last updated.

See Also

- » [About System Tables](#)

SYSVIEWS System Table

The SYSVIEWS table describes the view definitions within the current database.

The following table shows the contents of the SYSVIEWS system table.

SYSVIEWS system table

Column Name	Type	Length	Nullable	Contents
TABLEID	CHAR	36	NO	Unique identifier for the view (join with SYSTABLES . TABLEID)
VIEWDEFINITION	LONG VARCHAR	32,700	NO	Text of view definition
CHECKOPTION	CHAR	1	NO	'N' (check option not supported yet)
COMPILATIONSCHHEMAID	CHAR	36	NO	ID of the schema containing the view

See Also

- » [About System Tables](#)

Error Codes

This section contains descriptions of Splice Machine error codes, in these topics:

Error Class	Description
01	Warning Messages
07	Dynamic SQL Error
08	Connection Exception
0A	Feature not supported
0P	Invalid role specification
21	Cardinality Violation
22	Data Exception
23	Constraint Violation
24	Invalid Cursor State
25	Invalid Transaction State
28	Invalid Authorization Specification
2D	Invalid Transaction Termination
38	External Function Exception
39	External Routine Invocation Exception
3B	Invalid SAVEPOINT
40	Transaction Rollback
42	Syntax Error or Access Rule Violation
57	DRDA Network Protocol - Execution Failure
58	DRDA Network Protocol - Protocol Error
X0	Execution exceptions

XBCA	CacheService
XBCM	ClassManager
XBCX	Cryptography
XBM	Monitor
XCL	Execution exceptions
XCW	Upgrade unsupported
XCX	Internal Utility Errors
XY	Derby Property Exceptions
XCZ	org.apache.derby.database.UserUtility
XD00	Dependency Manager
XIE	Import/Export Exceptions
XJ	Connectivity Errors
XX	Security Exceptions
XN	Network Client Exceptions
XRE	Replication Exceptions
XSACI	Store - access.protocol.interface
XSAM	Store - AccessManager
XSAS	Store - Sort
XSAX	Store - access.protocol.XA statement
XSCB	Store - BTree
XSCG0	Conglomerate
XSCH	Heap
XSDA	RawStore - Data.Generic statement
XSDB	RawStore - Data.Generic transaction

XSDF	RawStore - Data.Filesystem statement
XSDG	RawStore - Data.Filesystem database
XSLA	RawStore - Log.Generic database exceptions
XSLB	RawStore - Log.Generic statement exceptions
XSRS	RawStore - protocol.Interface statement
XSTA2	XACT_TRANSACTION_ACTIVE
XSTB	RawStore - Transactions.Basic system
XXXXX	No SQLSTATE

Error Class 01: Warning Messages

Error Class 01: Warnings

SQLSTATE	Message Text
01001	An attempt to update or delete an already deleted row was made: No row was updated or deleted.
01003	Null values were eliminated from the argument of a column function.
01006	Privilege not revoked from user <authorizationID>.
01007	Role <authorizationID> not revoked from authentication id <authorizationID>.
01008	WITH ADMIN OPTION of role <authorizationID> not revoked from authentication id <authorizationID>.
01009	Generated column <columnName> dropped from table <tableName>.
0100E	XX Attempt to return too many result sets.
01500	The constraint <constraintName> on table <tableName> has been dropped.
01501	The view <viewName> has been dropped.
01502	The trigger <triggerName> on table <tableName> has been dropped.
01503	The column <columnName> on table <tableName> has been modified by adding a not null constraint.
01504	The new index is a duplicate of an existing index: <indexName>.
01505	The value <valueName> may be truncated.
01522	The newly defined synonym '<synonymName>' resolved to the object '<objectName>' which is currently undefined.
01J01	Database '<databaseName>' not created, connection made to existing database instead.
01J02	Scroll sensitive cursors are not currently implemented.
01J04	The class '<className>' for column '<columnName>' does not implement java.io.Serializable or java.sql.SQLData. Instances must implement one of these interfaces to allow them to be stored.
01J05	Database upgrade succeeded. The upgraded database is now ready for use. Revalidating stored prepared statements failed. See next exception for details of failure.
01J06	ResultSet not updatable. Query does not qualify to generate an updatable ResultSet.

SQLSTATE	Message Text
01J07	ResultSetHoldability restricted to ResultSet.CLOSE_CURSORS_AT_COMMIT for a global transaction.
01J08	Unable to open resultSet type <resultSetType>. ResultSet type <resultSetType> opened.
01J10	Scroll sensitive result sets are not supported by server; remapping to forward-only cursor
01J12	Unable to obtain message text from server. See the next exception. The stored procedure SYSIBM.SQLCAMESSAGE is not installed on the server. Please contact your database administrator.
01J13	Number of rows returned (<number>) is too large to fit in an integer; the value returned will be truncated.
01J14	SQL authorization is being used without first enabling authentication.
01J15	Your password will expire in <remainingDays> day(s). Please use the SYSCS_UTIL.SYSCS MODIFY_PASSWORD procedure to change your password in database '<databaseName>'.
01J16	Your password is stale. To protect the database, you should update your password soon. Please use the SYSCS_UTIL.SYSCS MODIFY_PASSWORD procedure to change your password in database '<databaseName>'.
01J17	Statistics are unavailable or out of date for one or more tables involved in this query.

Error Class 07: Dynamic SQL Errors

Error Class 07: Dynamic SQL Errors

SQLSTATE	Message Text
07000	At least one parameter to the current statement is uninitialized.
07004	Parameter <parameterName> is an <procedureName> procedure parameter and must be registered with CallableStatement.registerOutParameter before execution.
07009	No input parameters.

Error Class 08: Connection Exception

Error Class 08: Connection Exception

SQLSTATE	Message Text
08000	Connection closed by unknown interrupt.
08001.C.10	A connection could not be established because the security token is larger than the maximum allowed by the network protocol.
08001.C.11	A connection could not be established because the user id has a length of zero or is larger than the maximum allowed by the network protocol.
08001.C.12	A connection could not be established because the password has a length of zero or is larger than the maximum allowed by the network protocol.
08001.C.13	A connection could not be established because the external name (EXTNAM) has a length of zero or is larger than the maximum allowed by the network protocol.
08001.C.14	A connection could not be established because the server name (SRVNAM) has a length of zero or is larger than the maximum allowed by the network protocol.
08001.C.1	Required Splice DataSource property <propertyName> not set.
08001.C.2	<error> : Error connecting to server <serverName> on port <portNumber> with message <messageText>.
08001.C.3	SocketException: '<error>'
08001.C.4	Unable to open stream on socket: '<error>'.
08001.C.5	User id length (<number>) is outside the range of 1 to <number>.
08001.C.6	Password length (<value>) is outside the range of 1 to <number>.
08001.C.7	User id can not be null.
08001.C.8	Password can not be null.
08001.C.9	A connection could not be established because the database name '<databaseName>' is larger than the maximum length allowed by the network protocol.
08003	No current connection.
08003.C.1	getConnection() is not valid on a closed PooledConnection.

SQLSTATE	Message Text
08003.C.2	Lob method called after connection was closed
08003.C.3	The underlying physical connection is stale or closed.
08004	Connection refused : <connectionName>
08004.C.1	Connection authentication failure occurred. Reason: <reasonText>.
08004.C.2	The connection was refused because the database <databaseName> was not found.
08004.C.3	Database connection refused.
08004.C.4	User '<authorizationID>' cannot shut down database '<databaseName>'. Only the database owner can perform this operation.
08004.C.5	User '<authorizationID>' cannot (re)encrypt database '<databaseName>'. Only the database owner can perform this operation.
08004.C.6	User '<authorizationID>' cannot hard upgrade database '<databaseName>'. Only the database owner can perform this operation.
08004.C.7	Connection refused to database '<databaseName>' because it is in replication slave mode.
08004.C.8	User '<authorizationID>' cannot issue a replication operation on database '<databaseName>'. Only the database owner can perform this operation.
08004.C.9	Missing permission for user '<authorizationID>' to shutdown system [<exceptionMsg>].
08004.C.10	Cannot check system permission to create database '<databaseName>' [<exceptionMsg>].
08004.C.11	Missing permission for user '<authorizationID>' to create database '<databaseName>' [<exceptionMsg>].
08004.C.12	Connection authentication failure occurred. Either the supplied credentials were invalid, or the database uses a password encryption scheme not compatible with the strong password substitution security mechanism. If this error started after upgrade, refer to the release note for DERBY-4483 for options.
08004.C.13	Username or password is null or 0 length.
08006.C	A network protocol error was encountered and the connection has been terminated: <error>
08006.C.1	An error occurred during connect reset and the connection has been terminated. See chained exceptions for details.
08006.C.2	SocketException: '<error>'

SQLSTATE	Message Text
08006.C.3	A communications error has been detected: <error>.
08006.C.4	An error occurred during a deferred connect reset and the connection has been terminated. See chained exceptions for details.
08006.C.5	Insufficient data while reading from the network - expected a minimum of <number> bytes and received only <number> bytes. The connection has been terminated.
08006.C.6	Attempt to fully materialize lob data that is too large for the JVM. The connection has been terminated.
08006.C.8	com.splicemachine.db.jdbc.EmbeddedDriver is not registered with the JDBC driver manager
08006.C.9	Can't execute statement while in Restore Mode. Reboot database after restore operation is finished.
08006.D	Database '<databaseName>' shutdown.
08006.D.1	Database '<databaseName>' dropped.

Error Class 0A: Feature Not Supported

Error Class 0A: Feature Not Supported

SQLSTATE	Message Text
0A000.S	Feature not implemented: <featureName>.
0A000.SP	Feature not yet implemented in Splice Machine, but available soon: <featureName>.
0A000.C.6	The DRDA command <commandName> is not currently implemented. The connection has been terminated.
0A000.S.1	JDBC method is not yet implemented.
0A000.S.2	JDBC method <methodName> is not supported by the server. Please upgrade the server.
0A000.S.3	resultSetHoldability property <propertyName> not supported
0A000.S.4	cancel() not supported by the server.
0A000.S.5	Security mechanism '<mechanismName>' is not supported.
0A000.S.7	The data type '<datatypeName>' is not supported.

Error Class 0P: Invalid Role Specification

Error Class 0P: Invalid Role Specification

SQLSTATE	Message Text
0P000	Invalid role specification, role does not exist: '<roleName>'.
0P000.S.1	Invalid role specification, role not granted to current user or PUBLIC: '<roleName>'.

Error Class 21: Cardinality Violation

Error Class 21: Cardinality Violation

SQLSTATE	Message Text
21000	Scalar subquery is only allowed to return a single row.

Error Class 22: Data Exception

Error Class 22: Data Exception

SQLSTATE	Message Text
22001	A truncation error was encountered trying to shrink <value> '<value>' to length <value>.
22003	The resulting value is outside the range for the data type <datatypeName>.
22003.S.0	The modified row count was larger than can be held in an integer which is required by the JDBC spec. The real modified row count was <modifiedRowCount>.
22003.S.1	Year (<value>) exceeds the maximum '<value>'.
22003.S.2	Decimal may only be up to 31 digits.
22003.S.3	Overflow occurred during numeric data type conversion of '<datatypeName>' to <datatypeName>.
22003.S.4	The length (<number>) exceeds the maximum length (<datatypeName>) for the data type.
22005.S.1	Unable to convert a value of type '<typeName>' to type '<typeName>' : the encoding is not supported.
22005.S.2	The required character converter is not available.
22005.S.3	Unicode string cannot convert to Ebcdic string
22005.S.4	Unrecognized JDBC type. Type: <typeName>, columnCount: <value>, columnIndex: <value>.
22005.S.5	Invalid JDBC type for parameter <parameterName>.
22005.S.6	Unrecognized Java SQL type <datatypeName>.
22005.S.7	Unicode string cannot convert to UTF-8 string
22005	An attempt was made to get a data value of type '<datatypeName>' from a data value of type '<datatypeName>'.
22007.S.180	The string representation of a datetime value is out of range.
22007.S.181	The syntax of the string representation of a datetime value is incorrect.
22008.S	'<argument>' is an invalid argument to the <functionName> function.

SQLSTATE	Message Text
2200H.S	Sequence generator '<schemaName>.<sequenceName>' does not cycle. No more values can be obtained from this sequence generator.
2200L	Values assigned to XML columns must be well-formed DOCUMENT nodes.
2200M	Invalid XML DOCUMENT: <parserError>
2200V	Invalid context item for <operatorName> operator; context items must be well-formed DOCUMENT nodes.
2200W	XQuery serialization error: Attempted to serialize one or more top-level Attribute nodes.
22011	The second or third argument of the SUBSTR function is out of range.
22011.S.1	The range specified for the substring with offset <operatorName> and len <len> is out of range for the String: <str>.
22012	Attempt to divide by zero.
22013	Attempt to take the square root of a negative number, '<value>'.
22014	The start position for LOCATE is invalid; it must be a positive integer. The index to start the search from is '<startIndex>'. The string to search for is '<searchString>'. The string to search from is '<fromString>'.
22015	The '<functionName>' function is not allowed on the following set of types. First operand is of type '<typeName>'. Second operand is of type '<typeName>'. Third operand (start position) is of type '<typeName>'.
22018	Invalid character string format for type <typeName>.
22019	Invalid escape sequence, '<sequenceName>'. The escape string must be exactly one character. It cannot be a null or more than one character.
22020	Invalid trim string, '<string>'. The trim string must be exactly one character or NULL. It cannot be more than one character.
22021	Unknown character encoding '<typeName>'.
22025	Escape character must be followed by escape character, '_', or '%'. It cannot be followed by any other character or be at the end of the pattern.
22027	The built-in TRIM() function only supports a single trim character. The LTRIM() and RTRIM() built-in functions support multiple trim characters.
22028	The string exceeds the maximum length of <number>.

SQLSTATE	Message Text
22501	An ESCAPE clause of NULL returns undefined results and is not allowed.
2201X	Invalid row count for OFFSET, must be ≥ 0 .
2201Y	Invalid LEAD, LAG for OFFSET, must be greater or equal to 0 and less than Integer.MAX_VALUE. Got '<value>'.
2202A	Missing argument for first(), last() function.
2202B	Missing argument for lead(), lag() function.
2202C	"default" argument for lead(), lag() function is not implemented.
2202D	NULL value for data type <string> not supported.
2202E	A <string> column cannot be aggregated.
2201W	Row count for FIRST/NEXT/TOP must be ≥ 1 and row count for LIMIT must be ≥ 0 .
2201Z	NULL value not allowed for <string> argument.

Error Class 23: Constraint Violation

Error Class 23: Constraint Violation

SQLSTATE	Message Text
23502	Column '<columnName>' cannot accept a NULL value.
23503	<value> on table '<tableName>' caused a violation of foreign key constraint '<constraintName>' for key <keyName>. The statement has been rolled back.
23505	The statement was aborted because it would have caused a duplicate key value in a unique or primary key constraint or unique index identified by '<value>' defined on '<value>'.
23513	The check constraint '<constraintName>' was violated while performing an INSERT or UPDATE on table '<tableName>'.

Error Class 24: Invalid Cursor State

Error Class 24: Invalid Cursor State

SQLSTATE	Message Text
24 000	Invalid cursor state - no current row.
24501.S	The identified cursor is not open.

Error Class 25: Invalid Transaction State

Error Class 25: Invalid Transaction State

SQLSTATE	Message Text
25001	Cannot close a connection while a transaction is still active.
25001.S.1	Invalid transaction state: active SQL transaction.
25501	Unable to set the connection read-only property in an active transaction.
25502	An SQL data change is not permitted for a read-only connection, user or database.
25503	DDL is not permitted for a read-only connection, user or database.
25505	A read-only user or a user in a read-only database is not permitted to disable read-only mode on a connection.

Error Class 28: Invalid Authorization Specification

Error Class 28: Invalid Authorization Specification

SQLSTATE	Message Text
28502	The user name '<authorizationID>' is not valid.

Error Class 2D: Invalid Transaction Termination

Error Class 2D: Invalid Transaction Termination

SQLSTATE	Message Text
2D521.S.1	setAutoCommit(true) invalid during global transaction.
2D521.S.2	COMMIT or ROLLBACK invalid for application execution environment.

Error Class 38: External Function Exception

Error Class 38: External Function Exception

SQLSTATE	Message Text
38000	The exception '<exception>' was thrown while evaluating an expression.
38001	The external routine is not allowed to execute SQL statements.
38002	The routine attempted to modify data, but the routine was not defined as MODIFIES SQL DATA.
38004	The routine attempted to read data, but the routine was not defined as READS SQL DATA.

Error Class 39: External Routine Invocation Exception

Error Class 39: External Routine Invocation Exception

SQLSTATE	Message Text
39004	A NULL value cannot be passed to a method which takes a parameter of primitive type '<type>'.

Error Class 3B: Invalid SAVEPOINT

Error Class 3B: Invalid SAVEPOINT

SQLSTATE	Message Text
3B001.S	Savepoint <savepointName> does not exist or is not active in the current transaction.
3B002.S	The maximum number of savepoints has been reached.
3B501.S	A SAVEPOINT with the passed name already exists in the current transaction.
3B502.S	A RELEASE or ROLLBACK TO SAVEPOINT was specified, but the savepoint does not exist.

Error Class 40: Transaction Rollback

Error Class 40: Transaction Rollback

SQLSTATE	Message Text
40001	A lock could not be obtained due to a deadlock, cycle of locks and waiters is: <lockCycle>. The selected victim is XID : <transactionID>.
40XC0	Dead statement. This may be caused by catching a transaction severity error inside this statement.
40XD0	Container has been closed.
40XD1	Container was opened in read-only mode.
40XD2	Container <containerName> cannot be opened; it either has been dropped or does not exist.
40XL1	A lock could not be obtained within the time requested
40XL1.T.1	A lock could not be obtained within the time requested. The lockTable dump is: <tableDump>
40XT0	An internal error was identified by RawStore module.
40XT1	An exception was thrown during transaction commit.
40XT2	An exception was thrown during rollback of a SAVEPOINT.
40XT4	An attempt was made to close a transaction that was still active. The transaction has been aborted.
40XT5	Exception thrown during an internal transaction.
40XT6	Database is in quiescent state, cannot activate transaction. Please wait for a moment till it exits the quiescent state.
40XT7	Operation is not supported in an internal transaction.

Error Class 42: Syntax Error or Access Rule Violation

Error Class 42: Syntax Error or Access Rule Violation

SQLSTATE	Message Text
42000	Syntax error or access rule violation; see additional errors for details.
42500	User '<authorizationID>' does not have <permissionType> permission on table '<schemaName>'.'<tableName>'.
42501	User '<authorizationID>' does not have <permissionType> permission on table '<schemaName>'.'<tableName>' for grant.
42502	User '<authorizationID>' does not have <permissionType> permission on column '<columnName>' of table '<schemaName>'.'<tableName>'.
42503	User '<authorizationID>' does not have <permissionType> permission on column '<columnName>' of table '<schemaName>'.'<tableName>' for grant.
42504	User '<authorizationID>' does not have <permissionType> permission on <objectName>' '<schemaName>'.'<tableName>'.
42505	User '<authorizationID>' does not have <permissionType> permission on <objectName>' '<schemaName>'.'<tableName>' for grant.
42506	User '<authorizationID>' is not the owner of <objectName>' '<schemaName>'.'<tableName>'.
42507	User '<authorizationID>' can not perform the operation in schema '<schemaName>'.
42508	User '<authorizationID>' can not create schema '<schemaName>'. Only database owner could issue this statement.
42509	Specified grant or revoke operation is not allowed on object '<objectName>'.
4250A	User '<authorizationID>' does not have <permissionName> permission on object '<schemaName>'.'<objectName>'.
4250B	Invalid database authorization property '<value>=<value>'.
4250C	User(s) '<authorizationID>' must not be in both read-only and full-access authorization lists.
4250D	Repeated user(s) '<authorizationID>' in access list '<listName>';
4250E	Internal Error: invalid <authorizationID> id in statement permission list.
4251A	Statement <value> can only be issued by database owner.

SQLSTATE	Message Text
4251B	PUBLIC is reserved and cannot be used as a user identifier or role name.
4251C	Role <authorizationID> cannot be granted to <authorizationID> because this would create a circularity.
4251D	Only the database owner can perform this operation.
4251E	No one can view the '<tableName>'.'<columnName>' column.
4251F	You cannot drop the credentials of the database owner.
4251G	Please set derby.authentication.builtin.algorithm to a valid message digest algorithm. The current authentication scheme is too weak to be used by NATIVE authentication.
4251H	Invalid NATIVE authentication specification. Please set derby.authentication.provider to a value of the form NATIVE:\$credentialsDB or NATIVE:\$credentialsDB:LOCAL (at the system level).
4251I	Authentication cannot be performed because the credentials database '<databaseName>' does not exist.
4251J	The value for the property '<propertyName>' is formatted badly.
4251K	The first credentials created must be those of the DBO.
4251L	The derby.authentication.provider property specifies '<dbName>' as the name of the credentials database. This is not a valid name for a database.
4251M	User '<authorizationID>' does not have <permissionType> permission to analyze table '<schemaName>'.'<tableName>'.
42601	In an ALTER TABLE statement, the column '<columnName>' has been specified as NOT NULL and either the DEFAULT clause was not specified or was specified as DEFAULT NULL.
42601.S.372	ALTER TABLE statement cannot add an IDENTITY column to a table.
42605	The number of arguments for function '<functionName>' is incorrect.
42606	An invalid hexadecimal constant starting with '<number>' has been detected.
42610	All the arguments to the COALESCE/VALUE function cannot be parameters. The function needs at least one argument that is not a parameter.
42611	The length, precision, or scale attribute for column, or type mapping '<value>' is not valid.
42613	Multiple or conflicting keywords involving the '<clause>' clause are present.
42621	A check constraint or generated column that is defined with '<value>' is invalid.

SQLSTATE	Message Text
42622	The name '<name>' is too long. The maximum length is '<number>'.
42734	Name '<name>' specified in context '<context>' is not unique.
42802	The number of values assigned is not the same as the number of specified or implied columns.
42803	An expression containing the column '<columnName>' appears in the SELECT list and is not part of a GROUP BY clause.
42815.S.713	The replacement value for '<value>' is invalid.
42815.S.171	The data type, length or value of arguments '<value>' and '<value>' is incompatible.
42818	Comparisons between '<type>' and '<type>' are not supported. Types must be comparable. String types must also have matching collation. If collation does not match, a possible solution is to cast operands to force them to the default collation (e.g. SELECT tablename FROM sys.systables WHERE CAST(tablename AS VARCHAR(128)) = 'T1')
42820	The floating point literal '<string>' contains more than 30 characters.
42821	Columns of type '<type>' cannot hold values of type '<type>'.
42824	An operand of LIKE is not a string, or the first operand is not a column.
42831	'<columnName>' cannot be a column of a primary key or unique key because it can contain null values.
42831.S.1	'<columnName>' cannot be a column of a primary key because it can contain null values.
42834	SET NULL cannot be specified because FOREIGN KEY '<key>' cannot contain null values.
42837	ALTER TABLE '<tableName>' specified attributes for column '<columnName>' that are not compatible with the existing column.
42846	Cannot convert types '<type>' to '<type>'.
42877	A qualified column name '<columnName>' is not allowed in the ORDER BY clause.
42878	The ORDER BY clause of a SELECT UNION statement only supports unqualified column references and column position numbers. Other expressions are not currently supported.
42879	The ORDER BY clause may not contain column '<columnName>', since the query specifies DISTINCT and that column does not appear in the query result.
4287A	The ORDER BY clause may not specify an expression, since the query specifies DISTINCT.
4287B	In this context, the ORDER BY clause may only specify a column number.

SQLSTATE	Message Text
42884	No authorized routine named '<routineName>' of type '<type>' having compatible arguments was found.
42886	'<value>' parameter '<value>' requires a parameter marker '?'.
42894	DEFAULT value or IDENTITY attribute value is not valid for column '<columnName>'.
428C1	Only one identity column is allowed in a table.
428EK	The qualifier for a declared global temporary table name must be SESSION.
428C2	DELETE ROWS is not supported for ON '<txnMode>'.
428C3	Temporary table columns cannot be referenced by foreign keys.
428C4	Attempt to add temporary table, '<txnMode>', as a view dependency.
42903	Invalid use of an aggregate function.
42908	The CREATE VIEW statement does not include a column list.
42909	The CREATE TABLE statement does not include a column list.
42915	Foreign Key '<key>' is invalid because '<value>'.
42916	Synonym '<synonym2>' cannot be created for '<synonym1>' as it would result in a circular synonym chain.
42939	An object cannot be created with the schema name '<schemaName>'.
4293A	A role cannot be created with the name '<authorizationID>', the SYS prefix is reserved.
42962	Long column type column or parameter '<columnName>' not permitted in declared global temporary tables or procedure definitions.
42995	The requested function does not apply to global temporary tables.
42X01	Syntax error: <error>.
42X02	<value>.
42X03	Column name '<columnName>' is in more than one table in the FROM list.

SQLSTATE	Message Text
42X04	Column '<columnName>' is either not in any table in the FROM list or appears within a join specification and is outside the scope of the join specification or appears in a HAVING clause and is not in the GROUP BY list. If this is a CREATE or ALTER TABLE statement then '<columnName>' is not a column in the target table.
42X05	Table/View '<objectName>' does not exist.
42X06	Too many result columns specified for table '<tableName>'.
42X07	Null is only allowed in a VALUES clause within an INSERT statement.
42X08	The constructor for class '<className>' cannot be used as an external virtual table because the class does not implement '<constructorName>'.
42X09	The table or alias name '<tableName>' is used more than once in the FROM list.
42X10	'<tableName>' is not an exposed table name in the scope in which it appears.
42X12	Column name '<columnName>' appears more than once in the CREATE TABLE statement.
42X13	Column name '<columnName>' appears more than once times in the column list of an INSERT statement.
42X14	'<columnName>' is not a column in table or VTI '<value>'.
42X15	Column name '<columnName>' appears in a statement without a FROM list.
42X16	Column name '<columnName>' appears multiple times in the SET clause of an UPDATE statement.
42X17	In the Properties list of a FROM clause, the value '<value>' is not valid as a joinOrder specification. Only the values FIXED and UNFIXED are valid.
42X19	The WHERE or HAVING clause or CHECK CONSTRAINT definition is a '<value>' expression. It must be a BOOLEAN expression.
42X20	Syntax error; integer literal expected.
42X23	Cursor <cursorName> is not updatable.
42X24	Column <columnName> is referenced in the HAVING clause but is not in the GROUP BY list.
42X25	The '<functionName>' function is not allowed on the type.
42X26	The class '<className>' for column '<columnName>' does not exist or is inaccessible. This can happen if the class is not public.

SQLSTATE	Message Text
42X28	Delete table '<tableName>' is not target of cursor '<cursorName>'.
42X29	Update table '<tableName>' is not the target of cursor '<cursorName>'.
42X30	Cursor '<cursorName>' not found. Verify that autocommit is OFF.
42X31	Column '<columnName>' is not in the FOR UPDATE list of cursor '<cursorName>'.
42X32	The number of columns in the derived column list must match the number of columns in table '<tableName>'.
42X33	The derived column list contains a duplicate column name '<columnName>'.
42X34	There is a ? parameter in the select list. This is not allowed.
42X35	It is not allowed for both operands of '<value>' to be ? parameters.
42X36	The '<operator>' operator is not allowed to take a ? parameter as an operand.
42X37	The unary '<operator>' operator is not allowed on the '<type>' type.
42X38	'SELECT *' only allowed in EXISTS and NOT EXISTS subqueries.
42X39	Subquery is only allowed to return a single column.
42X40	A NOT statement has an operand that is not boolean . The operand of NOT must evaluate to TRUE, FALSE, or UNKNOWN.
42X41	In the Properties clause of a FROM list, the property '<propertyName>' is not valid (the property was being set to '<value>').
42X42	Correlation name not allowed for column '<columnName>' because it is part of the FOR UPDATE list.
42X43	The ResultSetMetaData returned for the class/object '<className>' was null. In order to use this class as an external virtual table, the ResultSetMetaData cannot be null.
42X44	Invalid length '<number>' in column specification.
42X45	<type> is an invalid type for argument number <value> of <value>.
42X46	There are multiple functions named '<functionName>'. Use the full signature or the specific name.
42X47	There are multiple procedures named '<procedureName>'. Use the full signature or the specific name.
42X48	Value '<value>' is not a valid precision for <value>.

SQLSTATE	Message Text
42X49	Value '<value>' is not a valid integer literal.
42X50	No method was found that matched the method call <methodName>. <value>(<value>), tried all combinations of object and primitive types and any possible type conversion for any parameters the method call may have. The method might exist but it is not public and/or static, or the parameter types are not method invocation convertible.
42X51	The class '<className>' does not exist or is inaccessible. This can happen if the class is not public.
42X52	Calling method ('<methodName>') using a receiver of the Java primitive type '<type>' is not allowed.
42X53	The LIKE predicate can only have 'CHAR' or 'VARCHAR' operands. Type '<type>' is not permitted.
42X54	The Java method '<methodName>' has a ? as a receiver. This is not allowed.
42X55	Table name '<tableName>' should be the same as '<value>'.
42X56	The number of columns in the view column list does not match the number of columns in the underlying query expression in the view definition for '<value>'.
42X57	The getColumnCount() for external virtual table '<tableName>' returned an invalid value '<value>'. Valid values are greater than or equal to 1.
42X58	The number of columns on the left and right sides of the <tableName> must be the same.
42X59	The number of columns in each VALUES constructor must be the same.
42X60	Invalid value '<value>' for insertMode property specified for table '<tableName>'.
42X61	Types '<type>' and '<type>' are not <value> compatible.
42X62	'<value>' is not allowed in the '<schemaName>' schema.
42X63	The USING clause did not return any results. No parameters can be set.
42X64	In the Properties list, the invalid value '<value>' was specified for the useStatistics property. The only valid values are TRUE or FALSE.
42X65	Index '<index>' does not exist.
42X66	Column name '<columnName>' appears more than once in the CREATE INDEX statement.
42X68	No field '<fieldName>' was found belonging to class '<className>'. It may be that the field exists, but it is not public, or that the class does not exist or is not public.

SQLSTATE	Message Text
42X69	It is not allowed to reference a field ('<fieldName>') using a referencing expression of the Java primitive type '<type>'.
42X70	The number of columns in the table column list does not match the number of columns in the underlying query expression in the table definition for '<value>'.
42X71	Invalid data type '<datatypeName>' for column '<columnName>'.
42X72	No static field '<fieldName>' was found belonging to class '<className>'. The field might exist, but it is not public and/or static, or the class does not exist or the class is not public.
42X73	Method resolution for signature <value>.<value>(<value>) was ambiguous. (No single maximally specific method.)
42X74	Invalid CALL statement syntax.
42X75	No constructor was found with the signature <value>(<value>). It may be that the parameter types are not method invocation convertible.
42X76	At least one column, '<columnName>', in the primary key being added is nullable. All columns in a primary key must be non-nullable.
42X77	Column position '<columnPosition>' is out of range for the query expression.
42X78	Column '<columnName>' is not in the result of the query expression.
42X79	Column name '<columnName>' appears more than once in the result of the query expression.
42X80	VALUES clause must contain at least one element. Empty elements are not allowed.
42X81	A query expression must return at least one column.
42X82	The USING clause returned more than one row. Only single-row ResultSets are permissible.
42X83	The constraints on column '<columnName>' require that it be both nullable and not nullable.
42X84	Index '<index>' was created to enforce constraint '<constraintName>'. It can only be dropped by dropping the constraint.
42X85	Constraint '<constraintName>' is required to be in the same schema as table '<tableName>'.
42X86	ALTER TABLE failed. There is no constraint '<constraintName>' on table '<tableName>'.
42X87	At least one result expression (THEN or ELSE) of the '<expression>' expression must not be a '?'.
42X88	A conditional has a non-Boolean operand. The operand of a conditional must evaluate to TRUE, FALSE, or UNKNOWN.

SQLSTATE	Message Text
42X89	Types '<type>' and '<type>' are not type compatible. Neither type is assignable to the other type.
42X90	More than one primary key constraint specified for table '<tableName>'.
42X91	Constraint name '<constraintName>' appears more than once in the CREATE TABLE statement.
42X92	Column name '<columnName>' appears more than once in a constraint's column list.
42X93	Table '<tableName>' contains a constraint definition with column '<columnName>' which is not in the table.
42X94	<value> '<value>' does not exist.
42X96	The database class path contains an unknown jar file '<fileName>'.
42X98	Parameters are not allowed in a VIEW definition.
42X99	Parameters are not allowed in a TABLE definition.
42XA0	The generation clause for column '<columnName>' has data type '<datatypeName>', which cannot be assigned to the column's declared data type.
42XA1	The generation clause for column '<columnName>' contains an aggregate. This is not allowed.
42XA2	'<value>' cannot appear in a GENERATION CLAUSE because it may return unreliable results.
42XA3	You may not override the value of generated column '<columnName>'.
42XA4	The generation clause for column '<columnName>' references other generated columns. This is not allowed.
42XA5	Routine '<routineName>' may issue SQL and therefore cannot appear in a GENERATION CLAUSE.
42XA6	'<columnName>' is a generated column. It cannot be part of a foreign key whose referential action for DELETE is SET NULL or SET DEFAULT, or whose referential action for UPDATE is CASCADE.
42XA7	'<columnName>' is a generated column. You cannot change its default value.
42XA8	You cannot rename '<columnName>' because it is referenced by the generation clause of column '<columnName>'.
42XA9	Column '<columnName>' needs an explicit datatype. The datatype can be omitted only for columns with generation clauses.

SQLSTATE	Message Text
42XAA	The NEW value of generated column '<columnName>' is mentioned in the BEFORE action of a trigger. This is not allowed.
42XAB	NOT NULL is allowed only if you explicitly declare a datatype.
42XAC	'INCREMENT BY' value can not be zero.
42XAE	'<argName>' value out of range of datatype '<datatypeName>'. Must be between '<minValue>' and '<maxValue>'.
42XAF	Invalid 'MINVALUE' value '<minValue>'. Must be smaller than 'MAXVALUE: <maxValue>'.
42XAG	Invalid 'START WITH' value '<startValue>'. Must be between '<minValue>' and '<maxValue>'.
42XAH	A NEXT VALUE FOR expression may not appear in many contexts, including WHERE, ON, HAVING, ORDER BY, DISTINCT, CASE, GENERATION, and AGGREGATE clauses as well as WINDOW functions and CHECK constraints.
42XAI	The statement references the following sequence more than once: '<sequenceName>'.
42XAJ	The CREATE SEQUENCE statement has a redundant '<clauseName>' clause.
42Y00	Class '<className>' does not implement com.splicemachine.db.iapi.db.AggregateDefinition and thus cannot be used as an aggregate expression.
42Y01	Constraint '<constraintName>' is invalid.
42Y03.S.0	'<statement>' is not recognized as a function or procedure.
42Y03.S.1	'<statement>' is not recognized as a procedure.
42Y03.S.2	'<statement>' is not recognized as a function.
42Y04	Cannot create a procedure or function with EXTERNAL NAME '<name>' because it is not a list separated by periods. The expected format is <full java path>. <method name>.
42Y05	There is no Foreign Key named '<key>'.
42Y07	Schema '<schemaName>' does not exist
42Y08	Foreign key constraints are not allowed on system tables.
42Y09	Void methods are only allowed within a CALL statement.
42Y10	A table constructor that is not in an INSERT statement has all ? parameters in one of its columns. For each column, at least one of the rows must have a non-parameter.

SQLSTATE	Message Text
42Y11	A join specification is required with the '<clauseName>' clause.
42Y12	The ON clause of a JOIN is a '<expressionType>' expression. It must be a BOOLEAN expression.
42Y13	Column name '<columnName>' appears more than once in the CREATE VIEW statement.
42Y16	No public static method '<methodName>' was found in class '<className>'. The method might exist, but it is not public, or it is not static.
42Y22	Aggregate <aggregateType> cannot operate on type <type>.
42Y23	Incorrect JDBC type info returned for column <columnName>.
42Y24	View '<viewName>' is not updatable. (Views are currently not updatable.)
42Y25	'<tableName>' is a system table. Users are not allowed to modify the contents of this table.
42Y26	Aggregates are not allowed in the GROUP BY list.
42Y27	Parameters are not allowed in the trigger action.
42Y29	The SELECT list of a non-grouped query contains at least one invalid expression. When the SELECT list contains at least one aggregate then all entries must be valid aggregate expressions.
42Y30	The SELECT list of a grouped query contains at least one invalid expression. If a SELECT list has a GROUP BY, the list may only contain valid grouping expressions and valid aggregate expressions.
42Y32	Aggregator class '<className>' for aggregate '<aggregateName>' on type <type> does not implement com.splicemachine.db.iapi.sql.execute.ExecAggregator.
42Y33	Aggregate <aggregateName> contains one or more aggregates.
42Y34	Column name '<columnName>' matches more than one result column in table '<tableName>'.
42Y35	Column reference '<reference>' is invalid. When the SELECT list contains at least one aggregate then all entries must be valid aggregate expressions.
42Y36	Column reference '<reference>' is invalid, or is part of an invalid expression. For a SELECT list with a GROUP BY, the columns and expressions being selected may only contain valid grouping expressions and valid aggregate expressions.
42Y37	'<value>' is a Java primitive and cannot be used with this operator.
42Y38	insertMode = replace is not permitted on an insert where the target table, '<tableName>', is referenced in the SELECT.

SQLSTATE	Message Text
42Y39	'<value>' may not appear in a CHECK CONSTRAINT definition because it may return non-deterministic results.
42Y40	'<value>' appears multiple times in the UPDATE OF column list for trigger '<triggerName>'.
42Y41	'<value>' cannot be directly invoked via EXECUTE STATEMENT because it is part of a trigger.
42Y42	Scale '<scaleValue>' is not a valid scale for a <value>.
42Y43	Scale '<scaleValue>' is not a valid scale with precision of '<precision>'.
42Y44	Invalid key '<key>' specified in the Properties list of a FROM list. The case-sensitive keys that are currently supported are '<key>'.
42Y45	VTI '<value>' cannot be bound because it is a special trigger VTI and this statement is not part of a trigger action or WHEN clause.
42Y46	Invalid Properties list in FROM list. There is no index '<index>' on table '<tableName>'.
42Y47	Invalid Properties list in FROM list. The hint useSpark needs (true/false) and does not support '<value>'.
42Y48	Invalid Properties list in FROM list. Either there is no named constraint '<constraintName>' on table '<tableName>' or the constraint does not have a backing index.
42Y49	Multiple values specified for property key '<key>'.
42Y50	Properties list for table '<tableName>' may contain values for index or for constraint but not both.
42Y55	'<value>' cannot be performed on '<value>' because it does not exist.
42Y56	Invalid join strategy '<strategyValue>' specified in Properties list on table '<tableName>'. The currently supported values for a join strategy are: <supportedStrategyNames>.
42Y58	NumberFormatException occurred when converting value '<value>' for optimizer override '<value>'.
42Y59	Invalid value, '<value>', specified for hashInitialCapacity override. Value must be greater than 0.
42Y60	Invalid value, '<value>', specified for hashLoadFactor override. Value must be greater than 0.0 and less than or equal to 1.0.
42Y61	Invalid value, '<value>', specified for hashMaxCapacity override. Value must be greater than 0.
42Y62	'<statement>' is not allowed on '<viewName>' because it is a view.

SQLSTATE	Message Text
42Y63	Hash join requires an optimizable equijoin predicate on a column in the selected index or heap. An optimizable equijoin predicate does not exist on any column in table or index '<index>'. Use the 'index' optimizer override to specify such an index or the heap on table '<tableName>'.
42Y64	bulkFetch value of '<value>' is invalid. The minimum value for bulkFetch is 1.
42Y65	bulkFetch is not permitted on '<joinType>' joins.
42Y66	bulkFetch is not permitted on updatable cursors.
42Y67	Schema '<schemaName>' cannot be dropped.
42Y69	No valid execution plan was found for this statement. This is usually because an infeasible join strategy was chosen, or because an index was chosen which prevents the chosen join strategy from being used.
42Y70	The user specified an illegal join order. This could be caused by a join column from an inner table being passed as a parameter to an external virtual table.
42Y71	System function or procedure '<procedureName>' cannot be dropped.
42Y82	System generated stored prepared statement '<statement>' that cannot be dropped using DROP STATEMENT. It is part of a trigger.
42Y83	An untyped null is not permitted as an argument to aggregate <aggregateName>. Please cast the null to a suitable type.
42Y84	'<value>' may not appear in a DEFAULT definition.
42Y85	The DEFAULT keyword is only allowed in a VALUES clause when the VALUES clause appears within an INSERT statement.
42Y90	FOR UPDATE is not permitted in this type of statement.
42Y91	The USING clause is not permitted in an EXECUTE STATEMENT for a trigger action.
42Y92	<triggerName> triggers may only reference <value> transition variables/tables.
42Y93	Illegal REFERENCING clause: only one name is permitted for each type of transition variable/table.
42Y94	An AND or OR has a non-boolean operand. The operands of AND and OR must evaluate to TRUE, FALSE, or UNKNOWN.
42Y95	The '<operatorName>' operator with a left operand type of '<operandType>' and a right operand type of '<operandType>' is not supported.

SQLSTATE	Message Text
42Y96	Invalid Sort Strategy: '<sortStrategy>'.
42Y97	Invalid escape character at line '<lineNumber>', column '<columnName>'.
42Z02	Multiple DISTINCT aggregates are not supported at this time.
42Z07	Aggregates are not permitted in the ON clause.
42Z08	Bulk insert replace is not permitted on '<value>' because it has an enabled trigger (<value>).
42Z15	Invalid type specified for column '<columnName>'. The type of a column may not be changed.
42Z16	Only columns of type VARCHAR, CLOB, and BLOB may have their length altered.
42Z17	Invalid length specified for column '<columnName>'. Length must be greater than the current column length.
42Z18	Column '<columnName>' is part of a foreign key constraint '<constraintName>'. To alter the length of this column, you should drop the constraint first, perform the ALTER TABLE, and then recreate the constraint.
42Z19	Column '<columnName>' is being referenced by at least one foreign key constraint '<constraintName>'. To alter the length of this column, you should drop referencing constraints, perform the ALTER TABLE and then recreate the constraints.
42Z20	Column '<columnName>' cannot be made nullable. It is part of a primary key or unique constraint, which cannot have any nullable columns.
42Z20.S.1	Column '<columnName>' cannot be made nullable. It is part of a primary key, which cannot have any nullable columns.
42Z21	Invalid increment specified for identity column '<columnName>'. Increment cannot be zero.
42Z22	Invalid type specified for identity column '<columnName>'. The only valid types for identity columns are BIGINT, INT and SMALLINT.
42Z23	Attempt to modify an identity column '<columnName>'.
42Z24	Overflow occurred in identity value for column '<columnName>' in table '<tableName>'.
42Z25	INTERNAL ERROR identity counter. Update was called without arguments with current value != NULL.
42Z26	A column, '<columnName>', with an identity default cannot be made nullable.
42Z27	A nullable column, '<columnName>', cannot be modified to have identity default.

SQLSTATE	Message Text
42Z50	INTERNAL ERROR: Unable to generate code for <value>.
42Z53	INTERNAL ERROR: Type of activation to generate for node choice <value> is unknown.
42Z60	<value> not allowed unless database property <propertyName> has value '<value>'.
42Z70	Binding directly to an XML value is not allowed; try using XMLPARSE.
42Z71	XML values are not allowed in top-level result sets; try using XMLSERIALIZE.
42Z72	Missing SQL/XML keyword(s) '<keywords>' at line <lineNumber>, column <columnNumber>.
42Z73	Invalid target type for XMLSERIALIZE: '<typeName>'.
42Z74	XML feature not supported: '<featureName>'.
42Z75	XML query expression must be a string literal.
42Z76	Multiple XML context items are not allowed.
42Z77	Context item must have type 'XML'; '<value>' is not allowed.
42Z79	Unable to determine the parameter type for XMLPARSE; try using a CAST.
42Z90	Class '<className>' does not return an updatable ResultSet.
42Z91	subquery
42Z92	repeatable read
42Z93	Constraints '<constraintName>' and '<constraintName>' have the same set of columns, which is not allowed.
42Z97	Renaming column '<columnName>' will cause check constraint '<constraintName>' to break.
42Z99	String or Hex literal cannot exceed 64K.
42Z9A	read uncommitted
42Z9B	The external virtual table interface does not support BLOB or CLOB columns. '<value>' column '<value>'.
42Z9D.S.1	Procedures that modify SQL data are not allowed in BEFORE triggers.
42Z9D	'<statement>' statements are not allowed in '<triggerName>' triggers.

SQLSTATE	Message Text
42Z9E	Constraint '<constraintName>' is not a <value> constraint.
42Z9F	Too many indexes (<index>) on the table <tableName>. The limit is <number>.
42ZA0	Statement too complex. Try rewriting the query to remove complexity. Eliminating many duplicate expressions or breaking up the query and storing interim results in a temporary table can often help resolve this error.
42ZA1	Invalid SQL in Batch: '<batch>'.
42ZA2	Operand of LIKE predicate with type <type> and collation <value> is not compatible with LIKE pattern operand with type <type> and collation <value>.
42ZA3	The table will have collation type <type> which is different than the collation of the schema <type> hence this operation is not supported .
42ZB1	Parameter style DERBY_JDBC_RESULT_SET is only allowed for table functions.
42ZB2	Table functions can only have parameter style DERBY_JDBC_RESULT_SET.
42ZB3	XML is not allowed as the datatype of a user-defined aggregate or of a column returned by a table function.
42ZB4	'<schemaName>.<functionName>' does not identify a table function.
42ZB5	Class '<className>' implements VTICosting but does not provide a public, no-arg constructor.
42ZB6	A scalar value is expected, not a row set returned by a table function.
42ZC0	Window '<>windowName>' is not defined.
42ZC1	Only one window is supported.
42ZC2	Window function is illegal in this context: '<clauseName>' clause
42ZC3	A user defined aggregate may not have the name of an aggregate defined by the SQL Standard or the name of a builtin Derby function having one argument: '<aggregateName>'
42ZC4	User defined aggregate '<schemaName>'.'<aggregateName>' is bound to external class '<className>'. The parameter types of that class could not be resolved.
42ZC6	User defined aggregate '<schemaName>'.'<aggregateName>' was declared to have this input Java type: '<javaDataType>'. This does not extend the following actual bounding input Java type: '<javaDataType>'.

SQLSTATE	Message Text
42ZC7	User defined aggregate '<schemaName>'.'<aggregateName>' was declared to have this return Java type: '<javaDataType>'. This does not extend the following actual bounding return Java type: '<javaDataType>'.
42ZC8	Implementing class '<className>' for user defined aggregate '<schemaName>'.'<aggregateName>' could not be instantiated or was malformed. Detailed message follows: <detailedMessage>
43001	The truncate function was provided a null operand.
43002	The truncate function was provided an operand which it does not know how to handle: '<operand>'. It requires a DATE, TIMESTAMP, INTEGER or DECIMAL type.
43003	The truncate function expects a right-side argument of type CHAR for an operand of type DATE or TIMESTAMP but got: '<truncValue>'.
43004	The truncate function expects a right-side argument of type INTEGER for an operand of type DECIMAL but got: '<truncValue>'.
43005	The truncate function got an invalid right-side trunc value for operand type DATE: '<truncValue>'.
43006	The truncate function got an unknown right-side trunc value for operand type '<operand>': '<truncValue>'. Acceptable values are: '<acceptableValues>'.
44001	<dateOrTimestamp>s cannot be multiplied or divided. The operation is undefined.
44002	<dateOrTimestamp>s cannot be added. The operation is undefined.
44003	Timestamp '<dateOrTimestamp>' is out of range (~ from 21 Sep 1677 00:12:44 GMT to 11 Apr 2262 23:47:16 GMT).

Error Class 57: DRDA Network Protocol: Execution Failure

Error Class 57: DRDA Network Protocol: Execution Failure

SQLSTATE	Message Text
57017.C	There is no available conversion for the source code page, <codePage>, to the target code page, <codePage>. The connection has been terminated.

Error Class 58: DRDA Network Protocol: Protocol Error

Error Class 58: DRDA Network Protocol: Protocol Error

SQLSTATE	Message Text
58009.C.10	Network protocol exception: only one of the VCM, VCS length can be greater than 0. The connection has been terminated.
58009.C.11	The connection was terminated because the encoding is not supported.
58009.C.12	Network protocol exception: actual code point, <value>, does not match expected code point, <value>. The connection has been terminated.
58009.C.13	Network protocol exception: DDM collection contains less than 4 bytes of data. The connection has been terminated.
58009.C.14	Network protocol exception: collection stack not empty at end of same id chain parse. The connection has been terminated.
58009.C.15	Network protocol exception: DSS length not 0 at end of same id chain parse. The connection has been terminated.
58009.C.16	Network protocol exception: DSS chained with same id at end of same id chain parse. The connection has been terminated.
58009.C.17	Network protocol exception: end of stream prematurely reached while reading InputStream, parameter #<value>. The connection has been terminated.
58009.C.18	Network protocol exception: invalid FDOCA LID. The connection has been terminated.
58009.C.19	Network protocol exception: SECTKN was not returned. The connection has been terminated.
58009.C.20	Network protocol exception: only one of NVCM, NVCS can be non-null. The connection has been terminated.
58009.C.21	Network protocol exception: SCLDTA length, <length>, is invalid for RDBNAM. The connection has been terminated.
58009.C.7	Network protocol exception: SCLDTA length, <length>, is invalid for RDBCOLID. The connection has been terminated.
58009.C.8	Network protocol exception: SCLDTA length, <length>, is invalid for PKGID. The connection has been terminated.
58009.C.9	Network protocol exception: PKGNAMCSN length, <length>, is invalid at SQLAM <value>. The connection has been terminated.

SQLSTATE	Message Text
58010.C	A network protocol error was encountered. A connection could not be established because the manager <value> at level <value> is not supported by the server.
58014.C	The DDM command 0x<value> is not supported. The connection has been terminated.
58015.C	The DDM object 0x<value> is not supported. The connection has been terminated.
58016.C	The DDM parameter 0x<value> is not supported. The connection has been terminated.
58017.C	The DDM parameter value 0x<value> is not supported. An input host variable may not be within the range the server supports. The connection has been terminated.

Error Class XBCA: CacheService

Error Class XBCA: CacheService

SQLSTATE	Message Text
XBCA0 .S	Cannot create new object with key <key> in <cache> cache. The object already exists in the cache.

Error Class XBCM: ClassManager

Error Class XBCM: ClassManager

SQLSTATE	Message Text
XBCM1 .S	Java linkage error thrown during load of generated class <className>.
XBCM2 .S	Cannot create an instance of generated class <className>.
XBCM3 .S	Method <methodName>() does not exist in generated class <className>.
XBCM4 .S	Java class file format limit(s) exceeded: <value> in generated class <className>.
XBCM5 .S	This operation requires that the JVM level be at least <jvmLevel>.

Error Class XBCX: Cryptography

Error Class XBCX: Cryptography

SQLSTATE	Message Text
XBCX0 .S	Exception from Cryptography provider. See next exception for details.
XBCX1 .S	Initializing cipher with illegal mode, must be either ENCRYPT or DECRYPT.
XBCX2 .S	Initializing cipher with a boot password that is too short. The password must be at least <number> characters long.
XBCX5 .S	Cannot change boot password to null.
XBCX6 .S	Cannot change boot password to a non-string serializable type.
XBCX7 .S	Wrong format for changing boot password. Format must be : old_boot_password, new_boot_password.
XBCX8 .S	Cannot change boot password for a non-encrypted database.
XBCX9 .S	Cannot change boot password for a read-only database.
XBCXA .S	Wrong boot password.
XBCXB .S	Bad encryption padding '<value>' or padding not specified. 'NoPadding' must be used.
XBCXC .S	Encryption algorithm '<algorithmName>' does not exist. Please check that the chosen provider '<providerName>' supports this algorithm.
XBCXD .S	The encryption algorithm cannot be changed after the database is created.
XBCXE .S	The encryption provider cannot be changed after the database is created.
XBCXF .S	The class '<className>' representing the encryption provider cannot be found.
XBCXG .S	The encryption provider '<providerName>' does not exist.
XBCXH .S	The encryptionAlgorithm '<algorithmName>' is not in the correct format. The correct format is algorithm/feedbackMode/NoPadding.
XBCXI .S	The feedback mode '<mode>' is not supported. Supported feedback modes are CBC, CFB, OFB and ECB.
XBCXJ .S	The application is using a version of the Java Cryptography Extension (JCE) earlier than 1.2.1. Please upgrade to JCE 1.2.1 and try the operation again.

SQLSTATE	Message Text
XBCXK.S	The given encryption key does not match the encryption key used when creating the database. Please ensure that you are using the correct encryption key and try again.
XBCXL.S	The verification process for the encryption key was not successful. This could have been caused by an error when accessing the appropriate file to do the verification process. See next exception for details.
XBCXM.S	The length of the external encryption key must be an even number.
XBCXN.S	The external encryption key contains one or more illegal characters. Allowed characters for a hexadecimal number are 0-9, a-f and A-F.
XBCXO.S	Cannot encrypt the database when there is a global transaction in the prepared state.
XBCXP.S	Cannot re-encrypt the database with a new boot password or an external encryption key when there is a global transaction in the prepared state.
XBCXQ.S	Cannot configure a read-only database for encryption.
XBCXR.S	Cannot re-encrypt a read-only database with a new boot password or an external encryption key .
XBCXS.S	Cannot configure a database for encryption, when database is in the log archive mode.
XBCXT.S	Cannot re-encrypt a database with a new boot password or an external encryption key, when database is in the log archive mode.
XBCXU.S	Encryption of an un-encrypted database failed: <failureMessage>
XBCXV.S	Encryption of an encrypted database with a new key or a new password failed: <failureMessage>
XBCXW.S	The message digest algorithm '<algorithmName>' is not supported by any of the available cryptography providers. Please install a cryptography provider that supports that algorithm, or specify another algorithm in the derby.authentication.builtin.algorithm property.

Error Class XBM: Monitor

Error Class XBM: Monitor

SQLSTATE	Message Text
XBM01.D	Startup failed due to an exception. See next exception for details.
XBM02.D	Startup failed due to missing functionality for <value>. Please ensure your classpath includes the correct Splice software.
XBM05.D	Startup failed due to missing product version information for <value>.
XBM06.D	Startup failed. An encrypted database cannot be accessed without the correct boot password.
XBM07.D	Startup failed. Boot password must be at least 8 bytes long.
XBM08.D	Could not instantiate <value> StorageFactory class <value>.
XBM0A.D	The database directory '<directoryName>' exists. However, it does not contain the expected '<servicePropertiesName>' file. Perhaps Splice was brought down in the middle of creating this database. You may want to delete this directory and try creating the database again.
XBM0B.D	Failed to edit/write service properties file: <errorMessage>
XBM0C.D	Missing privilege for operation '<operation>' on file '<path>': <errorMessage>
XBM0G.D	Failed to start encryption engine. Please make sure you are running Java 2 and have downloaded an encryption provider such as jce and put it in your class path.
XBM0H.D	Directory <directoryName> cannot be created.
XBM0I.D	Directory <directoryName> cannot be removed.
XBM0J.D	Directory <directoryName> already exists.
XBM0K.D	Unknown sub-protocol for database name <databaseName>.
XBM0L.D	Specified authentication scheme class <className> does implement the authentication interface <interfaceName>.
XBM0M.D	Error creating an instance of a class named '<className>'. This class name was the value of the derby.authentication.provider property and was expected to be the name of an application-supplied implementation of com.splicemachine.db.authentication.UserAuthenticator. The underlying problem was: <detail>
XBM0N.D	JDBC Driver registration with java.sql.DriverManager failed. See next exception for details.

SQLSTATE	Message Text
XBM0P.D	Service provider is read-only. Operation not permitted.
XBM0Q.D	File <fileName> not found. Please make sure that backup copy is the correct one and it is not corrupted.
XBM0R.D	Unable to remove File <fileName>.
XBM0S.D	Unable to rename file '<fileName>' to '<fileName>'
XBM0T.D	Ambiguous sub-protocol for database name <databaseName>.
XBM0U.S	No class was registered for identifier <identifierName>.
XBM0V.S	An exception was thrown while loading class <className> registered for identifier <identifierName>.
XBM0W.S	An exception was thrown while creating an instance of class <className3> registered for identifier <identifierName>.
XBM0X.D	Supplied territory description '<value>' is invalid, expecting In[_CO[_variant]] In=lower-case two-letter ISO-639 language code, CO=upper-case two-letter ISO-3166 country codes, see java.util.Locale.
XBM03.D	Supplied value '<value>' for collation attribute is invalid, expecting UCS_BASIC or TERRITORY_BASED.
XBM04.D	Collator support not available from the JVM for the database's locale '<value>'.
XBM0Y.D	Backup database directory <directoryName> not found. Please make sure that the specified backup path is right.
XBM0Z.D	Unable to copy file '<fileName>' to '<fileName>'. Please make sure that there is enough space and permissions are correct.

Error Class XCL: Execution exceptions

Error Class XCL: Execution exceptions

SQLSTATE	Message Text
XCL01.S	Result set does not return rows. Operation <operationName> not permitted.
XCL05.S	Activation closed, operation <operationName> not permitted.
XCL07.S	Cursor '<cursorName>' is closed. Verify that autocommit is OFF.
XCL08.S	Cursor '<cursorName>' is not on a row.
XCL09.S	An Activation was passed to the '<methodName>' method that does not match the PreparedStatement.
XCL10.S	A PreparedStatement has been recompiled and the parameters have changed. If you are using JDBC you must prepare the statement again.
XCL12.S	An attempt was made to put a data value of type '<datatypeName>' into a data value of type '<datatypeName>'.
XCL13.S	The parameter position '<parameterPosition>' is out of range. The number of parameters for this prepared statement is '<number>'.
XCL14.S	The column position '<columnPosition>' is out of range. The number of columns for this ResultSet is '<number>'.
XCL15.S	A ClassCastException occurred when calling the compareTo() method on an object '<object>'. The parameter to compareTo() is of class '<className>'.
XCL16.S	ResultSet not open. Operation '<operation>' not permitted. Verify that autocommit is OFF.
XCL18.S	Stream or LOB value cannot be retrieved more than once
XCL19.S	Missing row in table '<tableName>' for key '<key>'.
XCL20.S	Catalogs at version level '<versionNumber>' cannot be upgraded to version level '<versionNumber>'.
XCL21.S	You are trying to execute a Data Definition statement (CREATE, DROP, or ALTER) while preparing a different statement. This is not allowed. It can happen if you execute a Data Definition statement from within a static initializer of a Java class that is being used from within a SQL statement.
XCL22.S	Parameter <parameterName> cannot be registered as an OUT parameter because it is an IN parameter.

SQLSTATE	Message Text
XCL23.S	SQL type number '<type>' is not a supported type by registerOutParameter().
XCL24.S	Parameter <parameterName> appears to be an output parameter, but it has not been so designated by registerOutParameter(). If it is not an output parameter, then it has to be set to type <type>.
XCL25.S	Parameter <parameterName> cannot be registered to be of type <type> because it maps to type <type> and they are incompatible.
XCL26.S	Parameter <parameterName> is not an output parameter.
XCL27.S	Return output parameters cannot be set.
XCL30.S	An IOException was thrown when reading a '<value>' from an InputStream.
XCL31.S	Statement closed.
XCL33.S	The table cannot be defined as a dependent of table <tableName> because of delete rule restrictions. (The relationship is self-referencing and a self-referencing relationship already exists with the SET NULL delete rule.)
XCL34.S	The table cannot be defined as a dependent of table <tableName> because of delete rule restrictions. (The relationship forms a cycle of two or more tables that cause the table to be delete-connected to itself (all other delete rules in the cycle would be CASCADE)).
XCL35.S	The table cannot be defined as a dependent of table <tableName> because of delete rule restrictions. (The relationship causes the table to be delete-connected to the indicated table through multiple relationships and the delete rule of the existing relationship is SET NULL.).
XCL36.S	The delete rule of foreign key must be <value>. (The referential constraint is self-referencing and an existing self-referencing constraint has the indicated delete rule (NO ACTION, RESTRICT or CASCADE).)
XCL37.S	The delete rule of foreign key must be <value>. (The referential constraint is self-referencing and the table is dependent in a relationship with a delete rule of CASCADE.)
XCL38.S	the delete rule of foreign key must be <ruleName>. (The relationship would cause the table to be delete-connected to the same table through multiple relationships and such relationships must have the same delete rule (NO ACTION, RESTRICT or CASCADE).)
XCL39.S	The delete rule of foreign key cannot be CASCADE. (A self-referencing constraint exists with a delete rule of SET NULL, NO ACTION or RESTRICT.)
XCL40.S	The delete rule of foreign key cannot be CASCADE. (The relationship would form a cycle that would cause a table to be delete-connected to itself. One of the existing delete rules in the cycle is not CASCADE, so this relationship may be definable if the delete rule is not CASCADE.)

SQLSTATE	Message Text
XCL41.S	the delete rule of foreign key can not be CASCADE. (The relationship would cause another table to be delete-connected to the same table through multiple paths with different delete rules or with delete rule equal to SET NULL.)
XCL42.S	CASCADE
XCL43.S	SET NULL
XCL44.S	RESTRICT
XCL45.S	NO ACTION
XCL46.S	SET DEFAULT
XCL47.S	Use of '<value>' requires database to be upgraded from version <versionNumber> to version <versionNumber> or later.
XCL48.S	TRUNCATE TABLE is not permitted on '<value>' because unique/primary key constraints on this table are referenced by enabled foreign key constraints from other tables.
XCL49.S	TRUNCATE TABLE is not permitted on '<value>' because it has an enabled DELETE trigger (<value>).
XCL50.S	Upgrading the database from a previous version is not supported. The database being accessed is at version level '<versionNumber>', this software is at version level '<versionNumber>'.
XCL51.S	The requested function can not reference tables in SESSION schema.
XCL52.S	The statement has been cancelled or timed out.

Error Class XCW: Upgrade unsupported

Error Class XCW: Upgrade unsupported

SQLSTATE	Message Text
XCW00 .D	Unsupported upgrade from '<value>' to '<value>'.

Error Class XCX: Internal Utility Errors

Error Class XCX: Internal Utility Errors

SQLSTATE	Message Text
XCXA0 .S	Invalid identifier: '<value>'.
XCXB0 .S	Invalid database classpath: '<classpath>'.
XCXC0 .S	Invalid id list.
XCXE0 .S	You are trying to do an operation that uses the territory of the database, but the database does not have a territory.

Error Class XCY: Splice Property Exceptions

Error Class XCY: Splice Property Exceptions

SQLSTATE	Message Text
XCY00.S	Invalid value for property '<value>'='<value>'.
XCY02.S	The requested property change is not supported '<value>'='<value>'.
XCY03.S	Required property '<propertyName>' has not been set.
XCY04.S	Invalid syntax for optimizer overrides. The syntax should be -- SPLICE-PROPERTIES propertyName = value [, propertyName = value]*
XCY05.S.2	Invalid setting of the derby.authentication.provider property. This property is already set to enable NATIVE authentication and cannot be changed.
XCY05.S.3	Invalid setting of the derby.authentication.provider property. To enable NATIVE authentication, use the SYSCS_UTIL.SYSCS_CREATE_USER procedure to store credentials for the database owner.

Error Class XCZ: com.splicemachine.db.database.UserUtility

Error Class XCZ:
com.splicemachine.db.database.UserUtility

SQLSTATE	Message Text
XCZ00.S	Unknown permission '<permissionName>'.
XCZ01.S	Unknown user '<authorizationName>'.
XCZ02.S	Invalid parameter '<value>'='<value>'.

Error Class XD00: Dependency Manager

Error Class XD00: Dependency Manager

SQLSTATE	Message Text
XD003.S	Unable to restore dependency from disk. DependableFinder = '<value>'. Further information: '<value>'.
XD004.S	Unable to store dependencies.

Error Class XIE: Import/Export Exceptions

Error Class XIE: Import/Export Exceptions

SQLSTATE	Message Text
XIE01.S	Connection was null.
XIE03.S	Data found on line <lineNumber> for column <columnName> after the stop delimiter.
XIE04.S	Data file not found: <fileName>
XIE05.S	Data file cannot be null.
XIE06.S	Entity name was null.
XIE07.S	Field and record separators cannot be substrings of each other.
XIE08.S	There is no column named: <columnName>.
XIE09.S	The total number of columns in the row is: <number>.
XIE0A.S	Number of columns in column definition, <columnName>, differ from those found in import file <type>.
XIE0B.S	Column '<columnName>' in the table is of type <type>, it is not supported by the import/export feature.
XIE0C.S	Illegal <delimiter> delimiter character '<character>'.
XIE0D.S	Cannot find the record separator on line <lineNumber>.
XIE0E.S	Read endOfFile at unexpected place on line <lineNumber>.
XIE0F.S	Character delimiter cannot be the same as the column delimiter.
XIE0I.S	An IOException occurred while writing data to the file.
XIE0J.S	A delimiter is not valid or is used more than once.
XIE0K.S	The period was specified as a character string delimiter.
XIE0M.S	Table '<tableName>' does not exist.
XIE0N.S	An invalid hexadecimal string '<hexString>' detected in the import file.
XIE0P.S	Lob data file <fileName> referenced in the import file not found.

SQLSTATE	Message Text
XIE0Q.S	Lob data file name cannot be null.
XIE0R.S	Import error on line <lineNumber> of file <fileName>: <details>
XIE10.S	Import error during reading source file <fileName> : <details>
XIE11.S	SuperCSVReader error during Import : <details>
XIE12.S	There was <details> RegionServer failures during a write with WAL disabled, the transaction has to rollback to avoid data loss.
XIE0S.S	The export operation was not performed, because the specified output file (<fileName>) already exists. Export processing will not overwrite an existing file, even if the process has permissions to write to that file, due to security concerns, and to avoid accidental file damage. Please either change the output file name in the export procedure arguments to specify a file which does not exist, or delete the existing file, then retry the export operation.
XIE0T.S	The export operation was not performed, because the specified large object auxiliary file (<fileName>) already exists. Export processing will not overwrite an existing file, even if the process has permissions to write to that file, due to security concerns, and to avoid accidental file damage. Please either change the large object auxiliary file name in the export procedure arguments to specify a file which does not exist, or delete the existing file, then retry the export operation.
XIE0U.S	The export operation was not performed, because the specified parameter (replicationCount) is less than or equal to zero.
XIE0X.S	The export operation was not performed, because value of the specified parameter (<paramName>) is wrong.

Error Class XJ: Connectivity Errors

Error Class XJ: Connectivity Errors

SQLSTATE	Message Text
XJ004.C	Database '<databaseName>' not found.
XJ008.S	Cannot rollback or release a savepoint when in auto-commit mode.
XJ009.S	Use of CallableStatement required for stored procedure call or use of output parameters: <value>
XJ010.S	Cannot issue savepoint when autoCommit is on.
XJ011.S	Cannot pass null for savepoint name.
XJ012.S	'<value>' already closed.
XJ013.S	No ID for named savepoints.
XJ014.S	No name for un-named savepoints.
XJ015.M	Splice system shutdown.
XJ016.S	Method '<methodName>' not allowed on prepared statement.
XJ017.S	No savepoint command allowed inside the trigger code.
XJ018.S	Column name cannot be null.
XJ020.S	Object type not convertible to TYPE '<typeName>', invalid java.sql.Types value, or object was null.
XJ021.S	Type is not supported.
XJ022.S	Unable to set stream: '<name>'.
XJ023.S	Input stream did not have exact amount of data as the requested length.
XJ025.S	Input stream cannot have negative length.
XJ028.C	The URL '<urlValue>' is not properly formed.
XJ030.S	Cannot set AUTOCOMMIT ON when in a nested connection.
XJ040.C	Failed to start database '<databaseName>' with class loader <classLoader>, see the next exception for details.

SQLSTATE	Message Text
XJ041.C	Failed to create database '<databaseName>', see the next exception for details.
XJ042.S	'<value>' is not a valid value for property '<propertyName>'.
XJ044.S	'<value>' is an invalid scale.
XJ045.S	Invalid or (currently) unsupported isolation level, '<levelName>', passed to Connection.setTransactionIsolation(). The currently supported values are java.sql.Connection.TRANSACTION_SERIALIZABLE, java.sql.Connection.TRANSACTION_REPEATABLE_READ, java.sql.Connection.TRANSACTION_READ_COMMITTED, and java.sql.Connection.TRANSACTION_READ_UNCOMMITTED.
XJ048.C	Conflicting boot attributes specified: <attributes>
XJ049.C	Conflicting create attributes specified.
XJ04B.S	Batch cannot contain a command that attempts to return a result set.
XJ04C.S	CallableStatement batch cannot contain output parameters.
XJ056.S	Cannot set AUTOCOMMIT ON when in an XA connection.
XJ057.S	Cannot commit a global transaction using the Connection, commit processing must go thru XAResource interface.
XJ058.S	Cannot rollback a global transaction using the Connection, commit processing must go thru XAResource interface.
XJ059.S	Cannot close a connection while a global transaction is still active.
XJ05B.C	JDBC attribute '<attributeName>' has an invalid value '<value>', valid values are '<value>'.
XJ05C.S	Cannot set holdability ResultSet.HOLD_CURSORS_OVER_COMMIT for a global transaction.
XJ061.S	The '<methodName>' method is only allowed on scroll cursors.
XJ062.S	Invalid parameter value '<value>' for ResultSet.setFetchSize(int rows).
XJ063.S	Invalid parameter value '<value>' for Statement.setMaxRows(int maxRows). Parameter value must be >= 0.
XJ064.S	Invalid parameter value '<value>' for setFetchDirection(int direction).
XJ065.S	Invalid parameter value '<value>' for Statement.setFetchSize(int rows).

SQLSTATE	Message Text
XJ066.S	Invalid parameter value '<value>' for Statement.setMaxFieldSize(int max).
XJ067.S	SQL text pointer is null.
XJ068.S	Only executeBatch and clearBatch allowed in the middle of a batch.
XJ069.S	No SetXXX methods allowed in case of USING execute statement.
XJ070.S	Negative or zero position argument '<argument>' passed in a Blob or Clob method.
XJ071.S	Negative length argument '<argument>' passed in a BLOB or CLOB method.
XJ072.S	Null pattern or searchStr passed in to a BLOB or CLOB position method.
XJ073.S	The data in this BLOB or CLOB is no longer available. The BLOB/CLOB's transaction may be committed, its connection closed or it has been freed.
XJ074.S	Invalid parameter value '<value>' for Statement.setQueryTimeout(int seconds).
XJ076.S	The position argument '<positionArgument>' exceeds the size of the BLOB/CLOB.
XJ077.S	Got an exception when trying to read the first byte/character of the BLOB/CLOB pattern using getBytes/getSubString.
XJ078.S	Offset '<value>' is either less than zero or is too large for the current BLOB/CLOB.
XJ079.S	The length specified '<number>' exceeds the size of the BLOB/CLOB.
XJ080.S	USING execute statement passed <number> parameters rather than <number>.
XJ081.C	Conflicting create/restore/recovery attributes specified.
XJ081.S	Invalid value '<value>' passed as parameter '<parameterName>' to method '<methodName>'
XJ085.S	Stream has already been read and end-of-file reached and cannot be re-used.
XJ086.S	This method cannot be invoked while the cursor is not on the insert row or if the concurrency of this ResultSet object is CONCUR_READ_ONLY.
XJ087.S	Sum of position('<pos>') and length('<length>') is greater than the size of the LOB plus one.
XJ088.S	Invalid operation: wasNull() called with no data retrieved.
XJ090.S	Invalid parameter: calendar is null.
XJ091.S	Invalid argument: parameter index <indexNumber> is not an OUT or INOUT parameter.

SQLSTATE	Message Text
XJ093.S	Length of BLOB/CLOB, <number>, is too large. The length cannot exceed <number>.
XJ095.S	An attempt to execute a privileged action failed.
XJ096.S	A resource bundle could not be found in the <packageName> package for <value>
XJ097.S	Cannot rollback or release a savepoint that was not created by this connection.
XJ098.S	The auto-generated keys value <value> is invalid
XJ099.S	The Reader/Stream object does not contain length characters
XJ100.S	The scale supplied by the registerOutParameter method does not match with the setter method. Possible loss of precision!
XJ103.S	Table name can not be null
XJ104.S	Shared key length is invalid: <value>.
XJ105.S	DES key has the wrong length, expected length <number>, got length <number>.
XJ106.S	No such padding
XJ107.S	Bad Padding
XJ108.S	Illegal Block Size
XJ110.S	Primary table name can not be null
XJ111.S	Foreign table name can not be null
XJ112.S	Security exception encountered, see next exception for details.
XJ113.S	Unable to open file <fileName> : <error>
XJ114.S	Invalid cursor name '<cursorName>'
XJ115.S	Unable to open resultSet with requested holdability <value>.
XJ116.S	No more than <number> commands may be added to a single batch.
XJ117.S	Batching of queries not allowed by J2EE compliance.
XJ118.S	Query batch requested on a non-query statement.
XJ121.S	Invalid operation at current cursor position.

SQLSTATE	Message Text
XJ122.S	No updateXXX methods were called on this row.
XJ123.S	This method must be called to update values in the current row or the insert row.
XJ124.S	Column not updatable.
XJ125.S	This method should only be called on ResultSet objects that are scrollable (type TYPE_SCROLL_INSENSITIVE).
XJ126.S	This method should not be called on sensitive dynamic cursors.
XJ128.S	Unable to unwrap for '<value>'.
XJ200.S	Exceeded maximum number of sections <value>
XJ202.S	Invalid cursor name '<cursorName>'.
XJ203.S	Cursor name '<cursorName>' is already in use
XJ204.S	Unable to open result set with requested holdability <holdValue>.
XJ206.S	SQL text '<value>' has no tokens.
XJ207.S	executeQuery method can not be used for update.
XJ208.S	Non-atomic batch failure. The batch was submitted, but at least one exception occurred on an individual member of the batch. Use getNextException() to retrieve the exceptions for specific batched elements.
XJ209.S	The required stored procedure is not installed on the server.
XJ210.S	The load module name for the stored procedure on the server is not found.
XJ211.S	Non-recoverable chain-breaking exception occurred during batch processing. The batch is terminated non-atomically.
XJ212.S	Invalid attribute syntax: <attributeSyntax>
XJ213.C	The traceLevel connection property does not have a valid format for a number.
XJ214.S	An IO Error occurred when calling free() on a CLOB or BLOB.
XJ215.S	You cannot invoke other java.sql.Clob/java.sql.Blob methods after calling the free() method or after the Blob/Clob's transaction has been committed or rolled back.

SQLSTATE	Message Text
XJ216.S	The length of this BLOB/CLOB is not available yet. When a BLOB or CLOB is accessed as a stream, the length is not available until the entire stream has been processed.
XJ217.S	The locator that was supplied for this CLOB/BLOB is invalid

Error Class XK: Security Exceptions

Error Class XK: Security Exceptions

SQLSTATE	Message Text
XK000.S	The security policy could not be reloaded: <reason>
XK001.S	Username not found in SYS.SYSUSERS.

Error Class XN: Network Client Exceptions

Error Class XN: Network Client Exceptions

SQLSTATE	Message Text
XN001.S	Connection reset is not allowed when inside a unit of work.
XN008.S	Query processing has been terminated due to an error on the server.
XN009.S	Error obtaining length of BLOB/CLOB object, exception follows.
XN010.S	Procedure name can not be null.
XN011.S	Procedure name length <number> is not within the valid range of 1 to <number>.
XN012.S	On <operatingSystemName> platforms, XA supports version <versionNumber> and above, this is version <versionNumber>
XN013.S	Invalid scroll orientation.
XN014.S	Encountered an Exception while reading from the stream specified by parameter #<value>. The remaining data expected by the server has been filled with 0x0. The Exception had this message: <messageText>.
XN015.S	Network protocol error: the specified size of the InputStream, parameter #<value>, is less than the actual InputStream length.
XN016.S	Encountered an Exception while trying to verify the length of the stream specified by parameter #<value>. The Exception had this message: <messageText>.
XN017.S	End of stream prematurely reached while reading the stream specified by parameter #<value>. The remaining data expected by the server has been filled with 0x0.
XN018.S	Network protocol error: the specified size of the Reader, parameter #<value>, is less than the actual InputStream length.
XN019.S	Error executing a <value>, server returned <value>.
XN020.S	Error marshalling or unmarshalling a user defined type: <messageDetail>
XN021.S	An object of type <sourceClassName> cannot be cast to an object of type <targetClassName>.

Error Class XRE: Replication Exceptions

Error Class XRE: Replication Exceptions

SQLSTATE	Message Text
XRE00	This LogFactory module does not support replicatiosn.
XRE01	The log received from the master is corrupted.
XRE02	Master and Slave at different versions. Unable to proceed with Replication.
XRE03	Unexpected replication error. See derby.log for details.
XRE04.C.1	Could not establish a connection to the peer of the replicated database '<dbname>' on address '<hostname>:<portname>'.
XRE04.C.2	Connection lost for replicated database '<dbname>'.
XRE05.C	The log files on the master and slave are not in synch for replicated database '<dbname>'. The master log instant is <masterfile>:<masteroffset>, whereas the slave log instant is <slavefile>:<slaveoffset>. This is FATAL for replication - replication will be stopped.
XRE06	The connection attempts to the replication slave for the database <dbname> exceeded the specified timeout period.
XRE07	Could not perform operation because the database is not in replication master mode.
XRE08	Replication slave mode started successfully for database '<dbname>'. Connection refused because the database is in replication slave mode.
XRE09.C	Cannot start replication slave mode for database '<dbname>'. The database has already been booted.
XRE10	Conflicting attributes specified. See reference manual for attributes allowed in combination with replication attribute '<attribute>'.
XRE11.C	Could not perform operation '<command>' because the database '<dbname>' has not been booted.
XRE12	Replication network protocol error for database '<dbname>'. Expected message type '<expectedtype>', but received type '<expectedtype>'.
XRE20.D	Failover performed successfully for database '<dbname>', the database has been shutdown.
XRE21.C	Error occurred while performing failover for database '<dbname>', Failover attempt was aborted.
XRE22.C	Replication master has already been booted for database '<dbname>'

SQLSTATE	Message Text
XRE23	Replication master cannot be started since unlogged operations are in progress, unfreeze to allow unlogged operations to complete and restart replication
XRE40	Could not perform operation because the database is not in replication slave mode.
XRE41.C	Replication operation 'failover' or 'stopSlave' refused on the slave database because the connection with the master is working. Issue the 'failover' or 'stopMaster' operation on the master database instead.
XRE42.C	Replicated database '<dbname>' shutdown.
XRE43	Unexpected error when trying to stop replication slave mode. To stop replication slave mode, use operation 'stopSlave' or 'failover'.

Error Class XSAI: Store - access.protocol.interface

Error Class XSAI: Store - access.protocol.interface

SQLSTATE	Message Text
XSAI2.S	The conglomerate (<value>) requested does not exist.
XSAI3.S	Feature not implemented.

Error Class XSAM: Store - AccessManager

Error Class XSAM: Store - AccessManager

SQLSTATE	Message Text
XSAM0 . S	Exception encountered while trying to boot module for '<value>'.
XSAM2 . S	There is no index or conglomerate with conglom id '<conglomID>' to drop.
XSAM3 . S	There is no index or conglomerate with conglom id '<conglomID>'.
XSAM4 . S	There is no sort called '<sortName>'.
XSAM5 . S	Scan must be opened and positioned by calling next() before making other calls.
XSAM6 . S	Record <recordNumber> on page <pageNumber> in container <containerName> not found.

Error Class XSAS: Store - Sort

Error Class XSAS: Store - Sort

SQLSTATE	Message Text
XSAS0.S	A scan controller interface method was called which is not appropriate for a scan on a sort.
XSAS1.S	An attempt was made to fetch a row before the beginning of a sort or after the end of a sort.
XSAS3.S	The type of a row inserted into a sort does not match the sort's template.
XSAS6.S	Could not acquire resources for sort.

Error Class XSAX: Store - access.protocol.XA statement

Error Class XSAX: Store - access.protocol.XA statement

SQLSTATE	Message Text
XSAX0.S	XA protocol violation.
XSAX1.S	An attempt was made to start a global transaction with an Xid of an existing global transaction.

Error Class XSCB: Store - BTree

Error Class XSCB: Store - BTree

SQLSTATE	Message Text
XSCB0.S	Could not create container.
XSCB1.S	Container <containerName> not found.
XSCB2.S	The required property <propertyName> not found in the property list given to createConglomerate() for a btree secondary index.
XSCB3.S	Unimplemented feature.
XSCB4.S	A method on a btree open scan has been called prior to positioning the scan on the first row (i.e. no next() call has been made yet). The current state of the scan is (<value>).
XSCB5.S	During logical undo of a btree insert or delete the row could not be found in the tree.
XSCB6.S	Limitation: Record of a btree secondary index cannot be updated or inserted due to lack of space on the page. Use the parameters derby.storage.pageSize and/or derby.storage.pageReservedSpace to work around this limitation.
XSCB7.S	An internal error was encountered during a btree scan - current_rh is null = <value>, position key is null = <value>.
XSCB8.S	The btree conglomerate <value> is closed.
XSCB9.S	Reserved for testing.

Error Class XSCG0: Conglomerate

Error Class XSCG0: Conglomerate

SQLSTATE	Message Text
XSCG0 .S	Could not create a template.

Error Class XSCH: Heap

Error Class XSCH: Heap

SQLSTATE	Message Text
XSCH0.S	Could not create container.
XSCH1.S	Container <containerName> not found.
XSCH4.S	Conglomerate could not be created.
XSCH5.S	In a base table there was a mismatch between the requested column number <number> and the maximum number of columns <number>.
XSCH6.S	The heap container with container id <containerID> is closed.
XSCH7.S	The scan is not positioned.
XSCH8.S	The feature is not implemented.

Error Class XSDA: RawStore - Data.Generic statement

Error Class XSDA: RawStore - Data.Generic statement

SQLSTATE	Message Text
XSDA1.S	An attempt was made to access an out of range slot on a page
XSDA2.S	An attempt was made to update a deleted record
XSDA3.S	Limitation: Record cannot be updated or inserted due to lack of space on the page. Use the parameters derby.storage.pageSize and/or derby.storage.pageReservedSpace to work around this limitation.
XSDA4.S	An unexpected exception was thrown
XSDA5.S	An attempt was made to undelete a record that is not deleted
XSDA6.S	Column <columnName> of row is null, it needs to be set to point to an object.
XSDA7.S	Restore of a serializable or SQLData object of class <className>, attempted to read more data than was originally stored
XSDA8.S	Exception during restore of a serializable or SQLData object of class <className>
XSDA9.S	Class not found during restore of a serializable or SQLData object of class <className>
XSDAA.S	Illegal time stamp <value>, either time stamp is from a different page or of incompatible implementation
XSDAB.S	cannot set a null time stamp
XSDAC.S	Attempt to move either rows or pages from one container to another.
XSDAD.S	Attempt to move zero rows from one page to another.
XSDAE.S	Can only make a record handle for special record handle id.
XSDAF.S	Using special record handle as if it were a normal record handle.
XSDAG.S	The allocation nested top transaction cannot open the container.
XSDAI.S	Page <page> being removed is already locked for deallocation.
XSDAJ.S	Exception during write of a serializable or SQLData object

SQLSTATE	Message Text
XSDAK.S	Wrong page is gotten for record handle <value>.
XSDAL.S	Record handle <value> unexpectedly points to overflow page.
XSDAM.S	Exception during restore of a SQLData object of class <className>. The specified class cannot be instantiated.
XSDAN.S	Exception during restore of a SQLData object of class <className>. The specified class encountered an illegal access exception.
XSDAO.S	Internal error: page <pageNumber> attempted latched twice.

Error Class XSDB: RawStore - Data.Generic transaction

Error Class XSDB: RawStore - Data.Generic transaction

SQLSTATE	Message Text
XSDB0.D	Unexpected exception on in-memory page <page>
XSDB1.D	Unknown page format at page <page>
XSDB2.D	Unknown container format at container <containerName> : <value>
XSDB3.D	Container information cannot change once written: was <value>, now <value>
XSDB4.D	Page <page> is at version <versionNumber>, the log file contains change version <versionNumber>, either there are log records of this page missing, or this page did not get written out to disk properly.
XSDB5.D	Log has change record on page <page>, which is beyond the end of the container.
XSDB6.D	Another instance of Splice may have already booted the database <databaseName>.
XSDB7.D	WARNING: Splice (instance <value>) is attempting to boot the database <databaseName> even though Splice (instance <value>) may still be active. Only one instance of Splice should boot a database at a time. Severe and non-recoverable corruption can result and may have already occurred.
XSDB8.D	WARNING: Splice (instance <value>) is attempting to boot the database <databaseName> even though Splice (instance <value>) may still be active. Only one instance of Splice should boot a database at a time. Severe and non-recoverable corruption can result if 2 instances of Splice boot on the same database at the same time. The derby.database.forceDatabaseLock=true property has been set, so the database will not boot until the db.lck is no longer present. Normally this file is removed when the first instance of Splice to boot on the database exits, but it may be left behind in some shutdowns. It will be necessary to remove the file by hand in that case. It is important to verify that no other VM is accessing the database before deleting the db.lck file by hand.
XSDB9.D	Stream container <containerName> is corrupt.
XSDBA.D	Attempt to allocate object <object> failed.
XSDBB.D	Unknown page format at page <page>, page dump follows: <value>
XSDBC.D	Write of container information to page 0 of container <container> failed. See nested error for more information.

Error Class XSDF: RawStore - Data.Filesystem statement

Error Class XSDF: RawStore - Data.Filesystem statement

SQLSTATE	Message Text
XSDF0.S	Could not create file <fileName> as it already exists.
XSDF1.S	Exception during creation of file <fileName> for container
XSDF2.S	Exception during creation of file <fileName> for container, file could not be removed. The exception was: <value>.
XSDF3.S	Cannot create segment <segmentName>.
XSDF4.S	Exception during remove of file <fileName> for dropped container, file could not be removed <value>.
XSDF6.S	Cannot find the allocation page <page>.
XSDF7.S	Newly created page failed to be latched <value>
XSDF8.S	Cannot find page <page> to reuse.
XSDFB.S	Operation not supported by a read only database
XSDFD.S	Different page image read on 2 I/Os on Page <page>, first image has incorrect checksum, second image has correct checksum. Page images follows: <value> <value>
XSdff.S	The requested operation failed due to an unexpected exception.
XSDFH.S	Cannot backup the database, got an I/O Exception while writing to the backup container file <fileName>.
XSDFI.S	Error encountered while trying to write data to disk during database recovery. Check that the database disk is not full. If it is then delete unnecessary files, and retry connecting to the database. It is also possible that the file system is read only, or the disk has failed, or some other problem with the media. System encountered error while processing page <page>.

Error Class XSDG: RawStore - Data.Filesystem database

Error Class XSDG: RawStore - Data.Filesystem database

SQLSTATE	Message Text
XSDG0.D	Page <page> could not be read from disk.
XSDG1.D	Page <page> could not be written to disk, please check if the disk is full, or if a file system limit, such as a quota or a maximum file size, has been reached.
XSDG2.D	Invalid checksum on Page <page>, expected=<value>, on-disk version=<value>, page dump follows: <value>
XSDG3.D	Meta-data for <containerName> could not be accessed to <type> <file>
XSDG5.D	Database is not in create mode when createFinished is called.
XSDG6.D	Data segment directory not found in <value> backup during restore. Please make sure that backup copy is the right one and it is not corrupted.
XSDG7.D	Directory <directoryName> could not be removed during restore. Please make sure that permissions are correct.
XSDG8.D	Unable to copy directory '<directoryName>' to '<value>' during restore. Please make sure that there is enough space and permissions are correct.
XSDG9.D	Splice thread received an interrupt during a disk I/O operation, please check your application for the source of the interrupt.

Error Class XSLA: RawStore - Log.Generic database exceptions

Error Class XSLA: RawStore - Log.Generic database exceptions

SQLSTATE	Message Text
XSLA0.D	Cannot flush the log file to disk <value>.
XSLA1.D	Log Record has been sent to the stream, but it cannot be applied to the store (Object <object>). This may cause recovery problems also.
XSLA2.D	System will shutdown, got I/O Exception while accessing log file.
XSLA3.D	Log Corrupted, has invalid data in the log stream.
XSLA4.D	Cannot write to the log, most likely the log is full. Please delete unnecessary files. It is also possible that the file system is read only, or the disk has failed, or some other problems with the media.
XSLA5.D	Cannot read log stream for some reason to rollback transaction <transactionID>.
XSLA6.D	Cannot recover the database.
XSLA7.D	Cannot redo operation <operation> in the log.
XSLA8.D	Cannot rollback transaction <value>, trying to compensate <value> operation with <value>
XSLAA.D	The store has been marked for shutdown by an earlier exception.
XSLAB.D	Cannot find log file <logfileName>, please make sure your logDevice property is properly set with the correct path separator for your platform.
XSLAC.D	Database at <value> have incompatible format with the current version of software, it may have been created by or upgraded by a later version.
XSLAD.D	log Record at instant <value> in log file <logfileName> corrupted. Expected log record length <value>, real length <value>.
XSLAE.D	Control file at <value> cannot be written or updated.
XSLAF.D	A Read Only database was created with dirty data buffers.
XSLAH.D	A Read Only database is being updated.
XSLAI.D	Cannot log the checkpoint log record
XSLAJ.D	The logging system has been marked to shut down due to an earlier problem and will not allow any more operations until the system shuts down and restarts.

SQLSTATE	Message Text
XSLAK.D	Database has exceeded largest log file number <value>.
XSLAL.D	log record size <value> exceeded the maximum allowable log file size <number>. Error encountered in log file <logfileName>, position <value>.
XSLAM.D	Cannot verify database format at <value> due to IOException.
XSLAN.D	Database at <value> has an incompatible format with the current version of the software. The database was created by or upgraded by version <versionNumber>.
XSLAO.D	Recovery failed unexpected problem <value>.
XSLAP.D	Database at <value> is at version <versionNumber>. Beta databases cannot be upgraded,
XSLAQ.D	cannot create log file at directory <directoryName>.
XSLAR.D	Unable to copy log file '<logfileName>' to '<value>' during restore. Please make sure that there is enough space and permissions are correct.
XSLAS.D	Log directory <directoryName> not found in backup during restore. Please make sure that backup copy is the correct one and it is not corrupted.
XSLAT.D	The log directory '<directoryName>' exists. The directory might belong to another database. Check that the location specified for the logDevice attribute is correct.

Error Class XSLB: RawStore - Log.Generic statement exceptions

Error Class XSLB: RawStore - Log.Generic statement exceptions

SQLSTATE	Message Text
XSLB1.S	Log operation <logOperation> encounters error writing itself out to the log stream, this could be caused by an errant log operation or internal log buffer full due to excessively large log operation.
XSLB2.S	Log operation <logOperation> logging excessive data, it filled up the internal log buffer.
XSLB4.S	Cannot find truncationLWM <value>.
XSLB5.S	Illegal truncationLWM instant <value> for truncation point <value>. Legal range is from <value> to <value>.
XSLB6.S	Trying to log a 0 or -ve length log Record.
XSLB8.S	Trying to reset a scan to <value>, beyond its limit of <value>.
XSLB9.S	Cannot issue any more change, log factory has been stopped.

Error Class XSRS: RawStore - protocol.Interface statement

Error Class XSRS: RawStore - protocol.Interface statement

SQLSTATE	Message Text
XSRS0.S	Cannot freeze the database after it is already frozen.
XSRS1.S	Cannot backup the database to <value>, which is not a directory.
XSRS4.S	Error renaming file (during backup) from <value> to <value>.
XSRS5.S	Error copying file (during backup) from <path> to <path>.
XSRS6.S	Cannot create backup directory <directoryName>.
XSRS7.S	Backup caught unexpected exception.
XSRS8.S	Log Device can only be set during database creation time, it cannot be changed on the fly.
XSRS9.S	Record <recordName> no longer exists
XSRSA.S	Cannot backup the database when unlogged operations are uncommitted. Please commit the transactions with backup blocking operations.
XSRSB.S	Backup cannot be performed in a transaction with uncommitted unlogged operations.
XSRSC.S	Cannot backup the database to <directoryLocation>, it is a database directory.
XSRSD.S	Database backup is disabled. Contact your Splice Machine representative to enable.
XSRSE.S	Unable to enable the enterprise Manager. Enterprise services are disabled. Contact your Splice Machine representative to enable.
XSRSF.S	LDAP authentication is disabled. Contact your Splice Machine representative to enable.
XSRSG.S	SpliceMachine Enterprise services are disabled and so will not run on an encrypted host. Contact your Splice Machine representative to enable.

Error Class XSTA2: XACT_TRANSACTION_ACTIVE

Error Class XSTA2: XACT_TRANSACTION_ACTIVE

SQLSTATE	Message Text
XSTA2.S	A transaction was already active, when attempt was made to make another transaction active.

Error Class XSTB: RawStore - Transactions.Basic system

Error Class XSTB: RawStore - Transactions.Basic system

SQLSTATE	Message Text
XSTB0.M	An exception was thrown during transaction abort.
XSTB2.M	Cannot log transaction changes, maybe trying to write to a read only database.
XSTB3.M	Cannot abort transaction because the log manager is null, probably due to an earlier error.
XSTB5.M	Creating database with logging disabled encountered unexpected problem.
XSTB6.M	Cannot substitute a transaction table with another while one is already in use.

Error Class XXXXX: No SQLSTATE

Error Class XXXXX: No SQLSTATE

SQLSTATE	Message Text
XXXXX	Normal database session close.

Error Class X0 - Execution exceptions

Error Class X0: Execution exceptions

SQLSTATE	Message Text
X0A00.S	The select list mentions column '<columnName>' twice. This is not allowed in queries with GROUP BY or HAVING clauses. Try aliasing one of the conflicting columns to a unique name.
X0X02.S	Table '<tableName>' cannot be locked in '<mode>' mode.
X0X03.S	Invalid transaction state - held cursor requires same isolation level
X0X05.S	Table/View '<tableName>' does not exist.
X0X07.S	Cannot remove jar file '<fileName>' because it is on your derby.database.classpath '<classpath>'.
X0X0D.S	Invalid column array length '<columnArrayLength>'. To return generated keys, column array must be of length 1 and contain only the identity column.
X0X0E.S	Table '<columnPosition>' does not have an auto-generated column at column position '<tableName>'.
X0X0F.S	Table '<columnName>' does not have an auto-generated column named '<tableName>'.
X0X10.S	The USING clause returned more than one row; only single-row ResultSets are permissible.
X0X11.S	The USING clause did not return any results so no parameters can be set.
X0X13.S	Jar file '<fileName>' does not exist in schema '<schemaName>'.
X0X14.S	The file '<fileName>' does not exist.
X0X57.S	An attempt was made to put a Java value of type '<type>' into a SQL value, but there is no corresponding SQL type. The Java value is probably the result of a method call or field access.
X0X60.S	A cursor with name '<cursorName>' already exists.
X0X61.S	The values for column '<columnName>' in index '<indexName>' and table '<schemaName>. <tableName>' do not match for row location <location>. The value in the index is '<value>', while the value in the base table is '<value>'. The full index key, including the row location, is '<indexKey>'. The suggested corrective action is to recreate the index.
X0X62.S	Inconsistency found between table '<tableName>' and index '<indexName>'. Error when trying to retrieve row location '<rowLocation>' from the table. The full index key, including the row location, is '<indexKey>'. The suggested corrective action is to recreate the index.
X0X63.S	Got IOException '<value>'.

SQLSTATE	Message Text
X0X67.S	Columns of type '<type>' may not be used in CREATE INDEX, ORDER BY, GROUP BY, UNION, INTERSECT, EXCEPT or DISTINCT statements because comparisons are not supported for that type.
X0X81.S	<value> '<value>' does not exist.
X0X85.S	Index '<indexName>' was not created because '<indexType>' is not a valid index type.
X0X86.S	0 is an invalid parameter value for ResultSet.absolute(int row).
X0X87.S	ResultSet.relative(int row) cannot be called when the cursor is not positioned on a row.
X0X95.S	Operation '<operationName>' cannot be performed on object '<objectName>' because there is an open ResultSet dependent on that object.
X0X99.S	Index '<indexName>' does not exist.
X0Y16.S	'<value>' is not a view. If it is a table, then use DROP TABLE instead.
X0Y23.S	Operation '<operationName>' cannot be performed on object '<objectName>' because VIEW '<viewName>' is dependent on that object.
X0Y24.S	Operation '<operationName>' cannot be performed on object '<objectName>' because STATEMENT '<statement>' is dependent on that object.
X0Y25.S	Operation '<operationName>' cannot be performed on object '<objectName>' because <value> '<value>' is dependent on that object.
X0Y26.S	Index '<indexName>' is required to be in the same schema as table '<tableName>'.
X0Y28.S	Index '<indexName>' cannot be created on system table '<tableName>'. Users cannot create indexes on system tables.
X0Y29.S	Operation '<operationName>' cannot be performed on object '<objectName>' because TABLE '<tableName>' is dependent on that object.
X0Y30.S	Operation '<operationName>' cannot be performed on object '<objectName>' because ROUTINE '<routineName>' is dependent on that object.
X0Y32.S	<value> '<value>' already exists in <value> '<value>'.
X0Y38.S	Cannot create index '<indexName>' because table '<tableName>' does not exist.
X0Y41.S	Constraint '<constraintName>' is invalid because the referenced table <tableName> has no primary key. Either add a primary key to <tableName> or explicitly specify the columns of a unique constraint that this foreign key references.

SQLSTATE	Message Text
X0Y42.S	Constraint '<constraintName>' is invalid: the types of the foreign key columns do not match the types of the referenced columns.
X0Y43.S	Constraint '<constraintName>' is invalid: the number of columns in <value> (<value>) does not match the number of columns in the referenced key (<value>).
X0Y44.S	Constraint '<constraintName>' is invalid: there is no unique or primary key constraint on table '<tableName>' that matches the number and types of the columns in the foreign key.
X0Y45.S	Foreign key constraint '<constraintName>' cannot be added to or enabled on table <tableName> because one or more foreign keys do not have matching referenced keys.
X0Y46.S	Constraint '<constraintName>' is invalid: referenced table <tableName> does not exist.
X0Y54.S	Schema '<schemaName>' cannot be dropped because it is not empty.
X0Y55.S	The number of rows in the base table does not match the number of rows in at least 1 of the indexes on the table. Index '<indexName>' on table '<schemaName>. <tableName>' has <number> rows, but the base table has <number> rows. The suggested corrective action is to recreate the index.
X0Y56.S	'<value>' is not allowed on the System table '<tableName>'.
X0Y57.S	A non-nullable column cannot be added to table '<tableName>' because the table contains at least one row. Non-nullable columns can only be added to empty tables.
X0Y58.S	Attempt to add a primary key constraint to table '<tableName>' failed because the table already has a constraint of that type. A table can only have a single primary key constraint.
X0Y59.S	Attempt to add or enable constraint(s) on table '<tableName>' failed because the table contains <rowName> row(s) that violate the following check constraint(s): <constraintName>.
X0Y63.S	The command on table '<tableName>' failed because null data was found in the primary key or unique constraint/index column(s). All columns in a primary or unique index key must not be null.
X0Y63.S.1	The command on table '<tableName>' failed because null data was found in the primary key/index column(s). All columns in a primary key must not be null.
X0Y66.S	Cannot issue commit in a nested connection when there is a pending operation in the parent connection.
X0Y67.S	Cannot issue rollback in a nested connection when there is a pending operation in the parent connection.
X0Y68.S	<value> '<value>' already exists.
X0Y69.S	<value> is not supported in trigger <triggerName>.

SQLSTATE	Message Text
X0Y70.S	INSERT, UPDATE and DELETE are not permitted on table <tableName> because trigger <triggerName> is active.
X0Y71.S	Transaction manipulation such as SET ISOLATION is not permitted because trigger <triggerName> is active.
X0Y72.S	Bulk insert replace is not permitted on '<value>' because it has an enabled trigger (<value>).
X0Y77.S	Cannot issue set transaction isolation statement on a global transaction that is in progress because it would have implicitly committed the global transaction.
X0Y78.S	Statement.executeQuery() cannot be called with a statement that returns a row count.
X0Y78.S.1	<value>.executeQuery() cannot be called because multiple result sets were returned. Use <value>.execute() to obtain multiple results.
X0Y78.S.2	<value>.executeQuery() was called but no result set was returned. Use <value>.executeUpdate() for non-queries.
X0Y79.S	Statement.executeUpdate() cannot be called with a statement that returns a ResultSet.
X0Y80.S	ALTER table '<tableName>' failed. Null data found in column '<columnName>'.
X0Y83.S	WARNING: While deleting a row from a table the index row for base table row <rowName> was not found in index with conglomerate id <id>. This problem has automatically been corrected as part of the delete operation.
X0Y84.T	Too much contention on sequence <sequenceName>. This is probably caused by an uncommitted scan of the SYS.SYSSEQUENCES catalog. Do not query this catalog directly. Instead, use the SYSCS_UTIL.SYSCS_PEEK_AT_SEQUENCE function to view the current value of a query generator.
X0Y85.S	The Splice property '<propertyName>' identifies a class which cannot be instantiated: '<className>'. See the next exception for details.
X0Y86.S	Splice could not obtain the locks needed to release the unused, preallocated values for the sequence '<schemaName>'.'<sequenceName>'. As a result, unexpected gaps may appear in this sequence.
X0Y87.S	There is already an aggregate or function with one argument whose name is '<schemaName>'.'<aggregateOrFunctionName>'.

Splice Machine Developer's Guide

This chapter describes how to do development work with your Splice Machine Database. It is divided into these sections:

- » [Developer Fundamentals](#)
- » [Functions and Stored Procedures](#)
- » [Tuning and Debugging] (#Tuning)

Developer Fundamentals

This section contains topics that contain in-depth information about fundamental aspects of working with Splice Machine:

[*Running Transactions](#) [*Working with Date and Time Values](#) [*Using Database Triggers](#) [*Using Foreign Keys](#) [*Using Window Functions](#) [*Using Temporary Tables](#) [*Using Spark Libraries](#) [*Using the Virtual Table Interface](#) [*Using External Tables](#) [*Using HCatalog](#) [*Connecting Through HAProxy](#) [*Using the MapReduce API](#) [*Working with HBase](#)

Functions and Stored Procedures

This section contains information about creating and using stored procedures and functions with Splice Machine, in these topics:

- » [Writing Functions and Stored Procedures](#)
- » [Storing/Updating Functions and Procs](#)
- » [Stored Procedure Examples](#)

Tuning and Debugging

This section contains information about tuning the performance of your database, as well as debugging slowdowns, in these topics:

- » [Introduction](#)
- » [Optimizing Queries](#)
- » [Using Statistics](#)
- » [Using Explain Plan](#)
- » [Explain Plan Examples](#)
- » [Logging](#)
- » [Debugging](#)
- » [Using Snapshots](#)

Splice Machine Fundamentals

This section contains the following fundamental topics for Splice Machine developers:

Topic	Describes
Importing Data Tutorial	How to import data into your Splice Machine database, using built-in procedures
Running Transactions	Introduces you to the basics of running transactions with Splice Machine.
Working with Dates	Provides an overview of working with date and time values in Splice Machine.
Using Database Triggers	Describes database triggers and how you can use them with Splice Machine.
Using Foreign Keys	Describes our implementation of foreign keys and how our implementation ensures referential integrity.
Using Window Functions	A quick summary of window functions, as implemented in Splice Machine SQL.
Using Temporary Tables	Describes how to use temporary tables with Splice Machine.
Roles and Authorization	Describes how Splice Machine authorizes which operations can be performed by which users.
Using Spark Libraries	Describes how to interface with Spark libraries in Splice Machine.
Using the VTI Interface	Allows you to use an SQL interface with data that is external to your database.
Using External Tables	Describes the use of external tables (which are stored as flat files outside of your database) with Splice Machine.
Using HCatalog	How to use Splice Machine with HCatalog.
Using MapReduce	The Splice Machine MapReduce API provides a simple programmatic interface for using MapReduce with HBase and taking advantage of the transactional capabilities that Splice Machine provides.

Topic	Describes
<u>Working with HBase</u>	Working in HBase with Splice Machine.

Running Transactions In Splice Machine

Splice Machine is a fully transactional database that supports ACID transactions. This allows you to perform actions such as *commit* and *rollback*; in a transactional context, this means that the database does not make changes visible to others until a *commit* has been issued.

This topic includes brief overview information about transaction processing with Splice Machine, in these sections:

- » [Transactions Overview](#)
 - » [ACID Transactions](#) describes what ACID transactions are and why they're important.
 - » [MVCC and Snapshot Isolation](#) describes what snapshot isolation is and how it works in Splice Machine.
- » [Using Transactions](#)
 - » [Committing and Rolling Back Transaction Changes](#) introduces autocommit, commit, and rollback of transactions.
 - » [A Simple Transaction Example](#) presents an example of a transaction using the `splice>` command line interface.
 - » [Using Savepoints](#) describes how to use savepoints within transactions.

Transactions Overview

A transaction is a unit of work performed in a database; to maintain the integrity of the database, each transaction must:

- » complete in its entirety or have no effect on the database
- » be isolated from other transactions that are running concurrently in the database
- » produce results that are consistent with existing constraints in the database
- » write its results to durable storage upon successful completion

ACID Transactions

The properties that describe how transactions must maintain integrity are Atomicity, Consistency, Isolation, and Durability. Transactions adhering to these properties are often referred to as *ACID* transactions. Here's a summary of ACID transaction properties:

Property	Description
<i>Atomicity</i>	Requires that each transaction be atomic, i.e. all-or-nothing: if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. Splice Machine guarantees atomicity in each and every situation, including power failures, errors, and crashes.
<i>Consistency</i>	Ensures that any transaction will bring the database from one valid state to another. Splice Machine makes sure that any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof.

Property	Description
<i>Isolation</i>	Ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially. Splice Machine implements snapshot isolation using MVCC to guarantee that this is true.
<i>Durability</i>	Ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. Splice Machine stores changes in durable storage when they are committed.

MVCC and Snapshot Isolation

Splice Machine employs a lockless *snapshot isolation design* that uses *Multiple Version Concurrency Control (MVCC)* to create a new version of the record every time it is updated and enforce consistency. Database systems use concurrency control systems to manage concurrent access. The simplest control method is to use locks that make sure that the writer is finished before any reader can proceed; however, this approach can be very slow. With snapshot isolation, each transaction has its own virtual snapshot of the database, which means that multiple transactions can operate concurrently without creating deadlock conditions.

When Splice Machine needs to update an item in the database, it doesn't actually overwrite the old data value. Instead, it creates a new version with a new timestamp. Which means that readers have access to the data that was available when they began reading, even if that data has been updated by a writer in the meantime. This is referred to as *point-in-time consistency* and ensures that:

- » Every transaction runs in its own *transactional context*, which includes a snapshot of the database from when the transaction began.
- » Every read made during a transaction will see a consistent snapshot of the database.
- » A transaction can only commit its changes if they do not conflict with updates that have been committed while the transaction was running.

Reading and Writing Database Values During a Transaction

When you begin a transaction, you start working within a *transactional context* that includes a snapshot of the database. The operations that read and write database values for your transaction modify your transactional context. When your transaction is complete, you can commit those modifications to the database. The commit of your transaction's changes succeeds unless a *write-write conflict* occurs, which happens when your transaction attempts to commit an update to a value, and another update to that value has already been committed by a transaction that started before your transaction.

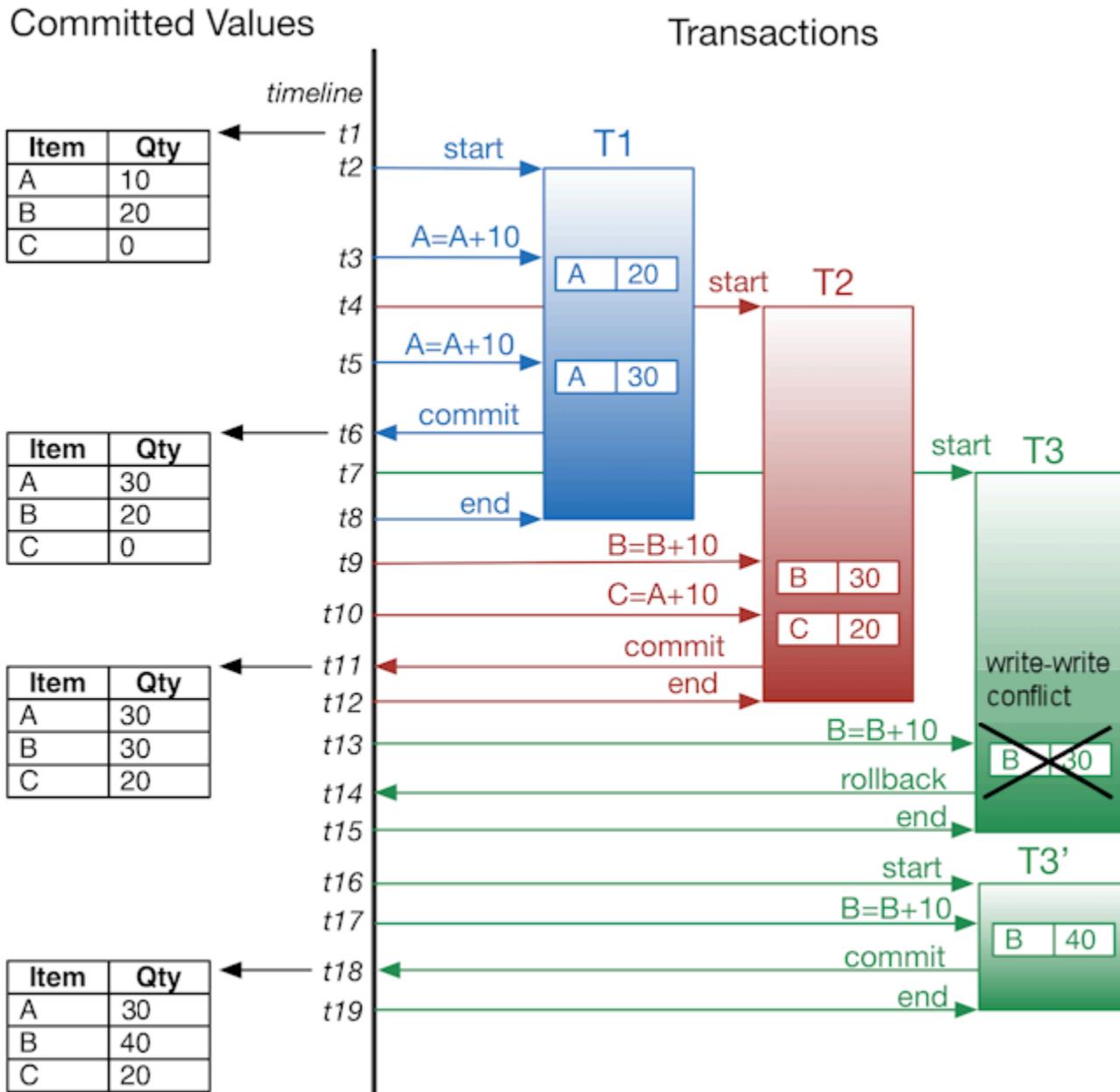
This means that the following statements are true with regard to reading values from and writing values to the database during a transaction:

- » When you read a value during a transaction, you get the value that was most recently set within your transactional context. If you've not already set the value within your context, then this is the value that had been most recently committed in the database before your transaction began (and before your transactional context was established).
- » When you write a value during a transaction, the value is set within your transactional context. It is only written to the database when you commit the transaction; that time is referred to as the *commit timestamp* for your transaction. The value changes that you commit then become visible to transactions that start after your transaction's commit timestamp (until another transaction modifies the value).
- » If two parallel transactions attempt to change the same value, then a *write-write conflict* occurs, and the commit of the

transaction that started later fails.

A Snapshot Isolation Example

The following diagram shows an example of snapshot isolation for a set of transactions, some of which are running in parallel:



Here's a tabular version of the same transactional timeline, showing the values committed in the database over time, with added commentary:

Time	Committed Values			Transactions				Comments
	A	B	C	T1	T2	T3	T3'	
t_1	10	20	0					
t_2				T1 Start				T1 starts. The starting values within its transactional context are: A=10, B=20, C=0.
t_3				A=A+10 [A=20]				T1 modifies the value of A within its context.
t_4					T2 Start			T2 starts. The starting values within its transactional context are the same as for T1: A=10, B=20, C=0.
t_5				A=A+10 [A=30]				T1 again modifies the value of A within its context
t_6	30	20	0	Commit				T1 commits its modifications to the database.
t_7						T3 Start		T3 starts. The starting values within its transactional context include the commits from T1: A=30, B=20, C=0.
t_8				T1 End				
t_9					B=B+10 [B=30]			T2 modifies the value of B within its context.
t_{10}					C=A+10 [C=20]			T2 modifies the value of C within its context; note that this computation correctly uses the value of A (10) that had been committed prior to the start of T2.
t_{11}	30	30	20		Commit			T2 commits its changes.
t_{12}					T2 End			
t_{13}						B=B+10 [B=30]		T3 modifies B; since its context includes the value of B before T2 committed, it modifies the original value of B [B=20] in its own context.

Time	Committed Values			Transactions				Comments
	A	B	C	T1	T2	T3	T3'	
t_{14}						Rollback		T3 attempts to commit its changes, which causes a <i>write-write conflict</i> , since T2 already committed an update to value B after T3 started. T3 rolls back and resets.
t_{15}						T3 End		
t_{16}							T3' Start	T3 reset (T3') starts. The starting values within its transactional context include the commits from T1 and T2: A=30, B=30, C=20.
t_{17}							B=B+10 [B=40]	T3 modifies the value of B, which has been updated and committed by T2.
t_{18}	40	40	20				Commit	T3 commits its changes.
t_{19}							T3' End	

Using Transactions

This section describes using transactions in your database, in these subsections:

- » [Committing and Rolling Back Transaction Changes](#) introduces autocommit, commit, and rollback of transactions.
- » [A Simple Transaction Example](#) presents an example of a transaction using the `splice>` command line interface.
- » [Using Savepoints](#) describes how to use savepoints within transactions.
- » [Using Rollback versus Rollback to Savepoint](#) discusses the differences between rolling back a transaction, and rolling back to a savepoint.

Committing and Rolling Back Transaction Changes

Within a transactional context, how the changes that you make are committed to the database depends on whether autocommit is enabled or disabled:

autocommit status	How changes are committed and rolled back
enabled	Changes are automatically committed whenever the operation completes successfully. If an operation reports any error, the changes are automatically rolled back.
disabled	Changes are only committed when you explicitly issue a <code>commit</code> command. Changes are rolled back when you explicitly issue a <code>rollback</code> command.



Autocommit is enabled by default. You typically disable autocommit when you want a block of operations to be committed atomically (all at once) instead of committing changes to the database after each operation.

You can turn `autocommit` on and off by issuing the `autocommit on` or `autocommit off` commands at the `splice>` prompt.

For more information, see these topics in the *Command Line Reference* section of this book:

- » [autocommit command](#)
- » [commit command](#)
- » [rollback command](#)

A Simple Transaction Example

Here is a simple example. Enter the following commands to see `commit` and `rollback` in action:

```

splice> create table myTbl (i int);
splice> autocommit off;                                - commits must be made explicitly
splice> insert into myTbl values 1,2,3; - inserted but not visible to others
splice> commit;                                     - now committed to the database
splice> select * from myTbl;                         - verify table contents
splice> insert into myTbl values 4,5; - insert more data
splice> select * from myTbl;                         - verify table contents
splice> rollback;                                    - roll back latest insertions
splice> select * from myTbl;                         - and verify again
...

```

You can turn `autocommit` back on by issuing the command: `autocommit on;`

Using Savepoints

Splice Machine supports the JDBC 3.0 Savepoint API, which adds methods for setting, releasing, and rolling back to savepoints within a transaction. Savepoints give you additional control over transactions by allowing you to define logical rollback points within a transaction, which effectively allows you to specify sub-transactions (also known as nested transactions).

You can specify multiple savepoints within a transaction. Savepoints are very useful when processing large transactions: you can implement error recovery schemes that allow you to rollback part of a transaction without having to abort the entire transaction.

You can use these commands to work with Savepoints:

- » create a savepoint with the `savepoint` command
- » release a savepoint with the `release savepoint` command
- » roll a transaction back to an earlier savepoint with the `rollback to savepoint` command

Example

First we'll create a table, turn autocommit off, and insert some data into the table. We then create a savepoint, and verify the contents of our table:

```
splice> CREATE TABLE myTbl(i int);
0 rows inserted/updated/deleted
splice> AUTOCOMMIT OFF;
splice> INSERT INTO myTbl VALUES 1,2,3;
3 rows inserted/updated/deleted
splice> SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3

3 rows selected
```

Next we add new values to the table and again verify its contents:

```
splice> INSERT INTO myTbl VALUES 4,5;
2 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3
4

55 rows selected
```

Now we roll back to our savepoint, and verify that the rollback worked:

```
splice> ROLLBACK TO SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3
3 rows selected
```

And finally, we commit the transaction:

```
COMMIT;
```

Using Rollback Versus Rollback to Savepoint

There's one important distinction you should be aware of between rolling back to a savepoint versus rolling back the entire transaction:

- » When you perform a `rollback`, Splice Machine aborts the entire transaction and creates a new transaction,
- » When you perform a `rollback to savepoint`, Splice Machine rolls back part of the changes, but does not create a new transaction.

Remember that this distinction also holds in a multi-tenant environment. In other words:

- » If two users are making modifications to the same table in separate transactions, and one user does a `rollback`, all changes made by that user prior to that rollback are no longer in the database.
- » Similarly, if two users are making modifications to the same table in separate transactions, and one user does a `rollback to savepoint`, all changes made by that user since the savepoint was established are no longer in the database.

See Also

- » [autocommit command](#)
- » [commit command](#)
- » [release savepoint command](#)
- » [rollback command](#)
- » [rollback to savepoint command](#)

» [savepoint command](#)

Working With Date and Time Values

This topic provides an overview of working with dates in Splice Machine.

For date and time values to work as expected in your database, you must make sure that all nodes in your cluster are set to the same time zone; otherwise the data you read from your database may differ, when you communicate with different servers!



Please contact your system administrator if you have any questions about this.

Date and Time Functions

Here is a summary of the `TIMESTAMP` functions included in this release of Splice Machine:

Function	Description
CURRENT_DATE	Returns the current date as a DATE value.
DATE	Returns a DATE value from a DATE value, a TIMESTAMP value, a string representation of a date or timestamp value, or a numeric value representing elapsed days since January 1, 1970.
DAY	Returns an integer value between 1 and 31 representing the day portion of a DATE value, a TIMESTAMP value, or a string representation of a date or timestamp value.
EXTRACT	Extracts various date and time components from a date expression.
LAST_DAY	Returns a DATE value representing the date of the last day of the month that contains the input date.
MONTH	Returns an integer value between 1 and 12 representing the month portion of a DATE value, a TIMESTAMP value, or a string representation of a date or timestamp value.
MONTH_BETWEEN	Returns a decimal number representing the number of months between two dates.
MONTHNAME	Returns the month name from a date expression.
NEXT_DAY	Returns the date of the next specified day of the week after a specified date.
NOW	Returns the current date and time as a TIMESTAMP value.
QUARTER	Returns the quarter number (1-4) from a date expression.
TIMESTAMP	Returns a timestamp value from a TIMESTAMP value, a string representation of a timestamp value, or a string of digits representing such a value.

Function	Description
TIMESTAMPADD	Adds the value of an interval to a TIMESTAMP value and returns the sum as a new timestamp.
TIMESTAMPDIFF	Finds the difference between two timestamps, in terms of the specified interval.
TO_CHAR	Returns string formed from a DATE value, using a format specification.
TO_DATE	Returns a DATE value formed from an input string representation, using a format specification.
WEEK	Returns the week number (1-53) from a date expression.
YEAR	Returns an integer value between 1 and 9999 representing the year portion of a DATE value, a TIMESTAMP value, or a string representation of a date or timestamp value.

Date Arithmetic

Splice Machine provides simple arithmetic operations addition and subtraction on date and timestamp values. You can:

- » find a future date value by adding an integer number of days to a date value
- » find a past date value by subtracting an integer number of days from a date value
- » subtract two date values to find the difference, in days, between those two values

Here's the syntax for these inline operations:

```

dateValue { "+" | "-" } numDays
| numDays '+' dateValue
| dateValue '-' dateValue

```

dateValue

A **TIMESTAMP** value. This can be a literal date value, a reference to a date value in a table, or the result of a function that produces a date value as its result.

numDays

An integer value expressing the number of days to add or subtract to a date value.

Result Types

The result type of adding or subtracting a number of days to/from a date value is a date value of the same type (**DATE** or **TIMESTAMP**) as the *dateValue* operand.

The result type of subtracting one date value from another is the number of days between the two dates. This can be a positive or negative integer value.

Notes

A few important notes about these operations:

- » Adding a number of days to a date value is commutative, which means that the order of the `dateValue` and `numDays` operands is irrelevant.
- » Subtraction of a number of days from a date value is not commutative: the left-side operand must be a date value.
- » Attempting to add two date values produces an error, as does attempting to use a date value in a multiplication or division operation.

Examples

This section presents several examples of using date arithmetic. We'll first set up a simple table that stores a string value, a DATE value, and a TIMESTAMP value, and we'll use those values in our example.

```
splice> CREATE TABLE date_math (s VARCHAR(30), d DATE, t TIMESTAMP);
0 rows inserted/updated/deleted

splice> INSERT INTO date_math values ('2012-05-23 12:24:36', '1988-12-26', '2000-06-07 17:12:30');
1 row inserted/updated/deleted
```

Example 1: Add a day to a date column and then to a timestamp column

```
splice> select d + 1 from date_math;
1
-----
1988-12-27
1 row selected

splice> select 1+t from date_math;
1
-----
2000-06-08 17:12:30.0
1 row selected
```

Example 2: Subtract a day from a timestamp column

```
splice> select t - 1 from date_math;
1
-----
2000-06-06 17:12:30.0
1 row selected
```

Example 3: Subtract a date column from the result of the `CURRENT_DATE` function

```
splice> select current_date - d from date_math;
1
-----
9551
1 row selected
```

Example 4: Additional examples using literal values

```
splice> values date('2011-12-26') + 1;
1
-----
2011-12-27
1 row selected

splice> values date('2011-12-26') - 1;
1
-----
2011-12-25
1 row selected

splice> values timestamp('2011-12-26', '17:13:30') + 1;
1
-----
2011-12-27 17:13:30.0
1 row selected

splice> values timestamp('2011-12-26', '17:13:30') - 1;
1
-----
2011-12-25 17:13:30.0
1 row selected

splice> values date('2011-12-26') - date('2011-06-05');
1
-----
204
1 row selected

splice> values date('2011-06-05') - date('2011-12-26');
1
-----
-204
1 row selected

splice> values timestamp('2015-06-07', '05:06:00') - current_date;
1
-----
108
1 row selected

splice> values timestamp('2011-06-05', '05:06:00') - date('2011-12-26');
1
-----
-203
1 row selected
```

See Also

All of the following are in the *SQL Reference Manual*:

- » [CURRENT_DATE](#)
- » [DATE](#)
- » [DATE](#)
- » [DAY](#)
- » [EXTRACT](#)
- » [LASTDAY](#)
- » [MONTH](#)
- » [MONTH_BETWEEN](#)
- » [MONTHNAME](#)
- » [NEXTDAY](#)
- » [NOW](#)
- » [QUARTER](#)
- » [TIME](#)
- » [TIMESTAMP](#)
- » [TO_CHAR](#)
- » [TO_DATE](#)
- » [WEEK](#)

Using Database Triggers

This topic describes database triggers and how you can use them with Splice Machine.

About Database Triggers

A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database. Triggers are mostly used for maintaining the integrity of the information on the database; they are most commonly used to:

- » automatically generate derived column values
- » enforce complex security authorizations
- » enforce referential integrity across nodes in a distributed database
- » enforce complex business rules
- » provide transparent event logging
- » provide sophisticated auditing
- » gather statistics on table access

Components of a Trigger

Each trigger has two required components:

Component	Description
Triggering event (or statement)	<p>The SQL statement that causes a trigger to be fired. This can be one of the following types of statement:</p> <ul style="list-style-type: none"> » INSERT » UPDATE » DELETE
Trigger action	<p>The procedure that contains the SQL statements to be executed when a triggering statement is issued and any trigger restrictions evaluate to TRUE.</p> <p>A trigger action is one of the following:</p> <ul style="list-style-type: none"> » arbitrary SQL » a call to a user-defined stored procedure

When a Trigger Fires

You can define both statement and row triggers as either *before triggers* or *after triggers*:

Trigger Type	Description
Before Triggers	A before trigger fires before the statement's changes are applied and before any constraints have been applied.
After Triggers	An after trigger fires after all constraints have been satisfied and after the changes have been applied to the target table.

How Often a Trigger Fires

You can define triggers as either *statement triggers* or *row triggers*, which defines how often a trigger will fire for a triggering event.

Trigger Type	Description
Statement Triggers	A statement trigger fires once per triggering event, regardless of how many rows (including zero rows) are modified by the event.
Row Triggers	A row trigger fires once for each row that is affected by the triggering event; for example, each row modified by an UPDATE statement. If no rows are affected by the event, the trigger does not fire.

NOTE: Triggers are statement triggers by default. You specify a row trigger in the FOR EACH clause of the `CREATE TRIGGER` statement.

Examples

This section presents examples of using database triggers.

Example 1: Row Level AFTER Trigger

This example shows a row level trigger that is called after a row is updated in the `employees` table. The action of this trigger is to insert one record into the audit trail table (`employees_log`) for each record that gets updated in the `employees` table.

```
CREATE TRIGGER log_salary_increase
AFTER UPDATE ON employees FOR EACH ROW
INSERT INTO employees_log
  (emp_id, log_date, new_salary, action)
VALUES (:new.empno, CURRENT_DATE, :new.salary, 'NEW SALARY');
```

If you then issue following statement to update salaries of all employees in the PD department:

```
UPDATE employees
SET salary = salary + 1000.0
WHERE department = 'PD';
```

Then the trigger will fire once (and one audit record will be inserted) for each employee in the department named PD.

Example 2: Statement Level After Trigger

This example shows a statement level trigger that is called after the `employees` table is updated. The action of this trigger is to insert exactly one record into the change history table (`reviews_history`) whenever the `employee_reviews` table is updated.

This example shows a row level trigger that is called after a row is updated in the `employees` table. The action of this trigger is to insert one record into the audit trail table (`employees_log`) for each record that gets updated in the `employees` table.

```
CREATE TRIGGER log_salary_increase
AFTER UPDATE ON employees referencing NEW as NEW FOR EACH ROW
INSERT INTO employees_log
  (emp_id, log_date, new_salary, action)
VALUES (NEW.empno, CURRENT_DATE, NEW.salary, 'NEW SALARY');
```

If you then issue the same Update statement as used in the previous example:

```
UPDATE employees SET salary = salary + 1000.0
WHERE department = 'PD';
```

Then the trigger will fire once and exactly one record will be inserted into the `employees_log` table, no matter how many records are updated by the statement.

Example 3: Statement Level Before Trigger

This example shows a row level trigger that is called before a row is inserted into the `employees` table.

```
CREATE TRIGGER empUpdateTrig
BEFORE UPDATE ON employees
  FOR EACH STATEMENT SELECT ID FROM myTbl;
```

See Also

- » [CREATE TRIGGER](#)
- » [DROP TRIGGER](#)
- » [Foreign keys](#)
- » [UPDATE](#)
- » [WHERE](#)

Foreign Keys and Referential Integrity

This topic describes the Splice Machine implementation of *foreign keys* and how our implementation ensures referential integrity.

See our *SQL Reference Manual* for full reference information about defining foreign keys using constraint clauses when creating a database table.

About Foreign Keys

A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables. A foreign key acts as a cross-reference between tables in that it references the primary key or unique key columns of another table, and thus establishes a link between them.

The purpose of a foreign key is to identify a particular row of the referenced table; as such, the foreign key must be equal to the key in some row of the primary table, or else be `null`. This rule is called a *referential integrity constraint between the two tables*, and is usually abbreviated as just *referential integrity*.

Maintaining Referential Integrity

To maintain referential integrity, Splice Machine ensures that database operations do not violate foreign key constraints, including not allowing any operations that will cause a foreign key to not correspond to a row in the referenced table. This can happen when a row is inserted, updated, or deleted in either table.

For example, suppose you have:

- » A table named `Players` with primary key `player_id`. This table is called the *parent table* or *referenced table*.
- » A second table named `PlayerStats` has a foreign key, which is also a column named `player_id`. This table is called the *child table* or *referencing table*.

The `player_id` column in the *referencing* table is the foreign key that references the primary key `player_id` in the *referenced* table.

When you insert a new record into the referencing `PlayerStats` table, the insertion must satisfy the foreign key constraint, which means that it must include a `player_id` value that is present in the referenced `Players` table. If this is not so, the insert operation fails in order to maintain the table's referential integrity.

About Foreign Key Constraints

You can define a foreign key constraint on a table when you create the table with the `CREATE TABLE` statement. Foreign key constraints are always immediate: a violation of a constraint immediately throws an exception.

Here's an example of defining a foreign key, in which we use the `REFERENCES` clause of a column definition in a `CREATE TABLE` statement:

```
CREATE TABLE t1 (c1 NUMERIC PRIMARY KEY);

CREATE TABLE t2 (
  c1 NUMERIC PRIMARY KEY,
  c2 NUMERIC REFERENCES t1(c1) );
```

And here's an example that uses the [CONSTRAINT](#) clause to name the foreign key constraint:

```
CREATE TABLE t3 (
  c1 NUMERIC,
  c2 NUMERIC,
  CONSTRAINT t1_fkey FOREIGN KEY (c1) REFERENCES t1);
```

You can also define a foreign key on a combination of columns:

```
CREATE TABLE dept_20
  (employee_id INT, hire_date DATE,
  CONSTRAINT fkey.empid.hiredate
  FOREIGN KEY (employee_id, hire_date)
  REFERENCES dept_21(employee_id, start_date));
```

See Also

- » [ALTER TABLE](#)
- » [CONSTRAINT](#)
- » [CREATE TABLE](#)
- » [Using Database Triggers](#)

Splice Machine Window Functions

An SQL *window function* performs a calculation across a set of table rows that are related to the current row, either by proximity in the table, or by the value of a specific column or set of columns; these columns are known as the *partition*.

This topic provides a very quick summary of window functions, as implemented in Splice Machine. For more general information about SQL window functions, we recommending visiting some of the sources listed in the [Additional Information](#) section at the end of this topic.

Here's a quick example of using a window function to operate on the following table:

OrderID	CustomerID	Amount
123	1	100
144	1	250
167	1	150
202	1	250
209	1	325
224	1	125
66	2	100
94	2	200
127	2	300
444	2	400

This query will find the first Order ID for each specified Customer ID in the above table:

```
SELECT OrderID, CustomerID,
       FIRST_VALUE(OrderID) OVER (
           PARTITION BY CustomerID
           ORDER BY OrderID
           ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW )
AS FirstOrderID
FROM ORDERS
WHERE CustomerID IN (1,2);
```

This works by partitioning (grouping) the selected rows by CustomerID, ordering them for purposes of applying the function to the rows in the partition, and then using the `FIRST_VALUE` window function to evaluate the OrderID values in each partition and find the first value in each. The results for our sample table are:

OrderID	CustomerID	FirstOrderID
123	1	123
144	1	123
167	1	123
202	1	123
209	1	123
224	1	123
66	2	66
94	2	66
127	2	66
444	2	66

See the [Window Frames](#) section below for a further explanation of this query.

About Window Functions

Window functions:

- » Operate on a window, or set of rows. The rows considered by a window function are produced by the query's `FROM` clause as filtered by its `WHERE`, `GROUP BY`, and `HAVING` clauses, if any. This means that any row that doesn't meet the `WHERE` condition is not seen by a window function.
- » Are similar to aggregate functions, except that a window function does not group rows into a single output row. Instead, a window function returns a value for every row in the window. This is sometimes referred to as tuple-based aggregation.
- » The values are calculated from the set of rows in the window.
- » Always contain an `OVER` clause, which determines how the rows of the query are divided and sequenced for processing by the window function.
- » The `OVER` clause can contain a `PARTITION` clause that specifies the set of rows in the table that form the window, relative to the current row.
- » The `OVER` clause can contain an optional `ORDER BY` clause that specifies in which order rows are processed by the window function. This `ORDER BY` clause is independent of the `ORDER BY` clause that specifies the order in which rows are output.

Note that the ROW NUMBER **must contain** an `ORDER BY` clause.

- » The OVER clause can also contain an optional *frame clause* that further restricts which of the rows in the partition are sent to the function for evaluation.

About Windows, Partitions, and Frames

Using window functions can seem complicated because they involve a number of overlapping terms, including *window*, *sliding window*, *partition*, *set*, and *window frame*. An additional complication is that window frames can be specified using either *rows* or *ranges*.

Let's start with basic terminology definitions:

Terms	Description
<i>window function</i>	A function that operates on a set of rows and produces output for each row.
<i>window partition</i>	<p>The grouping of rows within a table. Note that window partitions retains the rows, unlike aggregates,</p>
<i>window ordering</i>	The sequence of rows within each partition; this is the order in which the rows are passed to the window function for evaluation.
<i>window frame</i>	A frame of rows within a window partition, relative to the current row. The window frame is used to further restrict the set of rows operated on by a function, and is sometimes referred to as the <i>row</i> or <i>range</i> clause.
<i>OVER clause</i>	<p>This is the clause used to define how the rows of the table are divided, or partitioned, for processing by the window function. It also orders the rows within the partition. See the The OVER Clause section below for more information.</p>
<i>partitioning clause</i>	<p>An optional part of an OVER clause that divides the rows into partitions, similar to using the GROUP BY clause. The default partition is all rows in the table, though window functions are generally calculated over a partition. See the The Partition Clause section below for more information.</p>
<i>ordering clause</i>	<p>Defines the ordering of rows within each partition. See the The Order Clause section below for more information.</p>

Terms	Description
<i>frame clause</i>	<p>Further refines the set of rows when you include an ORDER BY clause in your window function specification, by allowing you to include or exclude rows or values within the ordering.</p> <p>See the The Frame Clause section below for examples and more information.</p>

The OVER Clause

A window function always contains an OVER clause, which determines how the rows of the query are divided, or partitioned, for processing by the window function.

```
expression OVER(
    [partitionClause]
    [orderClause]
    [frameClause] );
```

expression

Any value expression that does not itself contain window function calls.

NOTE: When you use an aggregate function such as AVG with an OVER clause, the aggregated value is computed per partition.

The Partition Clause

The partition clause, which is optional, specifies how the window function is broken down over groups, in the same way that GROUP BY specifies groupings for regular aggregate functions. Some example partitions are:

- » departments within an organization
- » regions within a geographic area
- » quarters within years for sales

NOTE: If you omit the partition clause, the default partition, which contains all rows in the table, is used. However, since window functions are used to perform calculations over subsets (partitions) of rows in a table, you generally should specify a partition clause.

Syntax

```
PARTITION BY expression [, ...]
```

expression [...]

A list of expressions that define the partitioning.

If you omit this clause, there is one partition that contains all rows in the entire table.

Here's a simple example of using the partition clause to compute the average order amount per customer:

```
SELECT OrderID, CustomerID, Amount,
       Avg(Amount) OVER (
           PARTITION BY CustomerID
           ORDER BY OrderID
           ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW )
AS AverageOrderAmt FROM ORDERS
WHERE CustomerID IN (1,2);
```

OrderID	CustomerID	Amount	AverageOrderAmt
123	1	100	200
144	1	250	200
167	1	150	200
202	1	250	200
209	1	325	200
224	1	125	200
66	2	100	250
94	2	200	250
127	2	300	250
444	2	400	250

The Order Clause

You can also control the order in which rows are processed by window functions using `ORDER BY` within your `OVER` clause. This is optional, though it is important for any ranking or cumulative functions.

Syntax

```
ORDER BY expression
  [ ASC | DESC | USING operator ]    [ NULLS FIRST | NULLS LAST ]
  [ , ... ]
```

Some notes about the `ORDER BY` clause in an `OVER` clause:

- » Ascending order (`ASC`) is the default ordering.

- » If you specify `NULLS LAST`, then `NULL` values are returned last; this is the default when you use `ASC` order.
- » If you specify `NULLS FIRST`, then `NULL` values are returned first; this is the default when you use `DESC` order.
- » The `ORDER BY` clause in your `OVER` clause *does not* have to match the order in which the rows are output.
- » You can only specify a *frame clause* if you include an `ORDER BY` clause in your `OVER` clause.

The Frame Clause

The optional frame clause defines which of the rows in the partition (the `frame`) should be evaluated by the window function. You can limit which rows in the partition are passed to the function in two ways:

- » Specify a `ROWS` frame to limit the frame to a fixed number of rows from the partition that precede or follow the current row.
- » Specify `RANGE` to only include rows in the frame whose evaluated value falls within a certain range of the current row's value. This is the default, and the current default range is 1, which means that only rows whose value matches that of the current row are passed to the function.

Some sources refer to the frame clause as the *Rows or Ranges* clause. If you omit this clause, the default is to include all rows

NOTE: Window frames can only be used when you include an `ORDER BY` clause within the `OVER` clause.

Syntax

This clause specifies two offsets: one determines the start of the window frame, and the other determines the end of the window frame.

```
[ RANGE | ROWS ] frameStart |
[RANGE | ROWS] BETWEEN frameStart AND frameEnd
```

RANGE

The frame includes rows whose values are within a specified range of the current row's value.

The range is determined by the `ORDER BY` column(s). Rows with identical values for their `ORDER BY` columns are referred to as *peer rows*.

ROWS

The frame includes a fixed number of rows based on their position in the table relative to the current row.

frameStart

Specifies the start of the frame.

For `ROWS` mode, you can specify:

```
UNBOUNDED PRECEDING
| value PRECEDING
| CURRENT ROW
| value FOLLOWING
```

value

A non-negative integer value.

For RANGE mode, you can only specify:

```
CURRENT ROW
| UNBOUNDED FOLLOWING
```

frameEnd

Specifies the end of the frame. The default value is CURRENT ROW.

For ROWS mode, you can specify:

```
value PRECEDING
| CURRENT ROW
| value FOLLOWING
| UNBOUNDED FOLLOWING
```

value

A non-negative integer value.

For RANGE mode, you can only specify:

```
CURRENT ROW
| UNBOUNDED FOLLOWING
```

Ranges and Rows

Probably the easiest way to understand how RANGE and ROWS work is by way of some simple OVER clause examples:

Example 1:

This clause can be used to apply a window function to all rows in the partition from the top of the partition to the current row:

```
OVER (PARTITION BY customerID ORDER BY orderDate)
```

Example 2:

Both of these clauses specify the same set of rows as [Example 1](#):

```
OVER (PARTITION BY customerID ORDER BY orderDate UNBOUNDED PRECEDING preceding)
```

```
OVER (PARTITION BY customerID ORDER BY orderDate RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
```

Example 3:

This clause can be used to apply a window function to the current row and the 3 preceding row's values in the partition:

```
OVER (PARTITION BY customerID ORDER BY orderDate ROWS 3 preceding)
```

FrameStart and FrameEnd

Some important notes about the frame clause:

- » UNBOUNDED PRECEDING means that the frame starts with the first row of the partition.
- » UNBOUNDED FOLLOWING means that the frame ends with the last row of the partition.
- » You must specify the *frameStart* first and the *frameEnd* last within the frame clause.
- » In ROWS mode, CURRENT ROW means that the frame starts or ends with the current row; in RANGE mode, CURRENT ROW means that the frame starts or ends with the current row's first or last peer in the ORDER BY ordering.
- » The default *frameClause* is to include all values from the start of the partition through the current row:

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

Common Frame Clauses

When learning about window functions, you may find references to these specific frame clause types:

Frame Clause Type	Example
<i>Recycled</i>	BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
<i>Cumulative</i>	BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
<i>Rolling</i>	BETWEEN 2 PRECEDING AND 2 FOLLOWING

Examples

This is a simple example that doesn't use a frame clause:

- Rank each year within a player by the number of home runs hit by that player:

```
RANK() OVER (PARTITION BY playerID ORDER BY H desc);
```

Here are some examples of window functions using frame clauses:

- Compute the running sum of G for each player:

```
SUM(G) OVER (PARTITION BY playerID ORDER BY yearID  
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW);
```

- Compute the career year:

```
YearID - min(YEARID) OVER (PARTITION BY playerID  
RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) + 1;
```

- Compute a rolling average of games by player:

```
AVG(G) OVER (PARTITION BY playerID ORDER BY yearID  
ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING);
```

The Ranking Functions

A subset of our window functions are known as *ranking functions*:

- » DENSE_RANK ranks each row in the result set. If values in the ranking column are the same, they receive the same rank. The next number in the ranking sequence is then used to rank the row or rows that follow, which means that DENSE_RANK always returns consecutive numbers.
- » RANK ranks each row in the result set. If values in the ranking column are the same, they receive the same rank. However, the next number in the ranking sequence is then skipped, which means that RANK can return non-consecutive numbers.
- » ROW NUMBER assigns a sequential number to each row in the result set.

All ranking functions **must include** an ORDER BY clause in the OVER() clause, since that is how they compute ranking values.

Window Function Restrictions

Because window functions are only allowed in `ORDER BY` clauses, and because window functions are computed after both `WHERE` and `HAVING`, you sometimes need to use subqueries with window functions to accomplish what seems like it could be done in a simpler query.

For example, because you cannot use an `OVER` clause in a `WHERE` clause, a query like the following is not possible:

```
SELECT *
FROM Batting
WHERE rank() OVER (PARTITION BY playerID ORDER BY G) = 1;
```

And because `WHERE` and `HAVING` are computed before the windowing functions, this won't work either:

```
SELECT playerID, rank() OVER (PARTITION BY playerID ORDER BY G) as player_rank FRO
M Batting
WHERE player_rank = 1;
```

Instead, you need to use a subquery:

```
SELECT *
FROM (
    SELECT playerID, G, rank() OVER (PARTITION BY playerID ORDER BY G) as "pos"
    FROM Batting
) tmp
WHERE "pos" = 1;
```

And note that the above subquery will add a rank column to the original columns,

Window Functions Included in This Release

Splice Machine is currently expanding the set of SQL functions already able to take advantage of windowing functionality.

The `OVER` clause topic completes the complete reference information for `OVER`.

Here is a list of the functions that currently support windowing:

- » [AVG](#)
- » [COUNT](#)
- » [DENSE_RANK](#)
- » [FIRST_VALUE](#)
- » [LAG](#)
- » [LAST_VALUE](#)
- » [LEAD](#)

- » MAX
- » MIN
- » RANK
- » ROW NUMBER
- » SUM

Additional Information

There are numerous articles about window functions that you can find online. Here are a few you might find valuable:

- » The [simple talk articles from Red Gate](#) about window functions are probably the most straightforward and comprehensive descriptions of window functions.
- » This [Oracle Technology Network article](#) provides an excellent technical introduction.
- » This [PostgreSQL page](#) introduces their version of window functions and links to other pages.
- » This [PostgreSQL wiki page](#) about SQL windowing queries page provides a succinct explanation of why windowing functions are used, and includes several useful examples.
- » The [Wikipedia SQL SELECT](#) page contains descriptions of specific window functions and links to other pages.

Using Temporary Database Tables

This topic describes how to use temporary tables with Splice Machine.

About Temporary Tables

You can use temporary tables when you want to temporarily save a result set for further processing. One common case for doing so is when you've constructed a result set by running multiple queries. You can also use temporary tables when performing complex queries that require extensive operations such as repeated multiple joins or sub-queries. Storing intermediate results in a temporary table can reduce overall processing time.

An example of using a temporary table to store intermediate results is a web-based application for travel reservations that allows customers to create several alternative itineraries, compare them, and then select one for purchase. Such an app could store each itinerary in a row in a temporary table, using table updates whenever the itinerary changes. When the customer decides upon a final itinerary, that temporary row is copied into a persistent table. And when the customer session ends, the temporary table is automatically dropped.

NOTE: Creating and operating with temporary tables does consume resources, and can affect performance of your queries.

Creating Temporary Tables

Splice Machine provides two statements you can use to create a temporary table; we provide multiple ways to create temporary tables to maintain compatibility with third party Business Intelligence tools.

Splice Machine does not currently support creating temporary tables stored as external tables.

Each of these statements creates the same kind of temporary table, using different syntax

Statement	Syntax
<code>CREATE TEMPORARY TABLE</code>	<pre>CREATE [LOCAL GLOBAL] TEMPORARY TABLE <i>table-name</i> { ({column-definition} Table-level constraint } [, {column-definition}] *) (column-name [, column-name] *) } [NOLOGGING ON COMMIT PRESERVE ROWS];</pre>

Statement	Syntax
<code>DECLARE GLOBAL TEMPORARY TABLE</code>	<pre>DECLARE GLOBAL TEMPORARY TABLE <i>table-Name</i> { <i>column-definition</i>[, <i>column-definition</i>] * } [ON COMMIT PRESERVE ROWS] [NOT LOGGED];</pre>

NOTE: Splice Machine generates a warning if you attempt to specify any other modifiers other than the NOLOGGING, NOT LOGGED, and ON COMMIT PRESERVE ROWS modifiers shown above.

Restrictions on Temporary Tables

You can use temporary tables just like you do permanently defined database tables, with several important exceptions and restrictions that are noted in this section, including these:

- » [Operational Limitations](#)
- » [Table Persistence](#)

Operational Limitations

Temporary tables have the following operational limitations; they:

- » exist only while a user session is alive
- » are visible in system tables, but are otherwise not visible to other sessions or transactions
- » cannot be altered using the `RENAME COLUMN` statements
- » do not get backed up
- » cannot be used as data providers to views
- » cannot be referenced by foreign keys in other tables
- » are not displayed by the `show tables` command

Also note that temporary tables persist across transactions in a session and are automatically dropped when a session terminates.

Table Persistence

Here are two important notes about temporary table persistence. Temporary tables:

- » persist across transactions in a session

- » are automatically dropped when a session terminates or expires
- » can also be dropped with the `DROP TABLE` statement

Example

```
create local temporary table temp_num_dt (
    smallint_col smallint not null,
    int_col int,
    primary key(smallint_col)) on commit preserve rows;
insert into temp_num_dt values (1,1);
insert into temp_num_dt values (3,2),(4,2),(5,null),(6,4),(7,8);
insert into temp_num_dt values (13,2),(14,2),(15,null),(16,null),(17,8);
select * from temp_num_dt;
```

See Also

- » [ALTER TABLE](#)
- » [CREATE TEMPORARY TABLE](#)
- » [DECLARE GLOBAL TEMPORARY TABLE](#)
- » [DROP TABLE](#)
- » [RENAME COLUMN](#)
- » [RENAME TABLE](#)

Using Spark Libraries with Splice Machine

One of the great features of Spark is that a large number of libraries have been and continue to be developed for use with Spark. This topic provides an example of interfacing to the Spark Machine Learning library (MLlib).

You can follow a similar path to interface with other Spark libraries, which involves these steps:

1. Create a class with an API that leverages functionality in the Spark library you want to use.
2. Write a custom procedure in your Splice Machine database that converts a Splice Machine result set into a Spark Resilient Distributed Dataset (RDD).
3. Use the Spark library with the RDD.

Example: Using Spark MLlib with Splice Machine Statistics

This section presents the sample code for interfacing Splice Machine with the Spark Machine Learning Library (MLlib), in these subsections:

- » [About the Splice Machine SparkMLibUtils Class API](#) describes the SparkMLibUtils class that Splice Machine provides for interfacing with this library.
- » [Creating our SparkStatistics Example Class](#) summarizes the `SparkStatistics` Java class that we created for this example.
- » [Run a Sample Program to Use Our Class](#) shows you how to define a custom procedure in your database to interface to the `SparkStatistics` class.

About the Splice Machine SparkMLibUtils Class API

Our example makes use of the Splice Machine `com.splicemachine.example.SparkMLibUtils` class, which you can use to interface between your Splice Machine database and the Spark Machine Learning library.

Here's are the public methods from the `SparkMLibUtils` class:

```
public static JavaRDDLocatedRow> resultSetToRDD(ResultSet rs)
    throws StandardException;

public static JavaRDDVector> locatedRowRDDToVectorRDD(JavaRDDLocatedRow> locatedRowJ
avaRDD, int[] fieldsToConvert)
    throws StandardException;

public static Vector convertExecRowToVector(ExecRow execRow, int[] fieldsToConvert)
    throws StandardException;

public static Vector convertExecRowToVector(ExecRow execRow)
    throws StandardException;
```

resultSetToRDD

Converts a Splice Machine result set into a Spark Resilient Distributed Dataset (RDD) object.

locatedRowRDDToVectorRDD

Transforms an RDD into a vector for use with the Machine Learning library. The `fieldsToConvert` parameter specifies which column positions to include in the vector.

convertExecRowToVector

Converts a Splice Machine execrow into a vector. The `fieldsToConvert` parameter specifies which column positions to include in the vector.

Creating our SparkStatistics Example Class

For this example, we define a Java class named `SparkStatistics` that can query a Splice Machine table, convert that results into a Spark JavaRDD, and then use the Spark MLlib to calculate statistics.

Our class, `SparkStatistics`, defines one public interface:

```
public class SparkStatistics {

    public static void getStatementStatistics(String statement, ResultSet[] resultSets) throws SQLException {
        try {
            // Run sql statement
            Connection con = DriverManager.getConnection("jdbc:default:connection");
            PreparedStatement ps = con.prepareStatement(statement);
            ResultSet rs = ps.executeQuery();

            // Convert result set to Java RDD
            JavaRDD<LocatedRow> resultSetRDD = ResultSetToRDD(rs);

            // Collect column statistics
            int[] fieldsToConvert = getFieldsToConvert(ps);
            MultivariateStatisticalSummary summary = getColumnStatisticsSummary(resultSetRDD, fieldsToConvert);

            IteratorNoPutResultSet resultsToWrap = wrapResults((EmbedConnection) con,
                getColumnStatistics(ps, summary, fieldsToConvert));
            resultSets[0] = new EmbedResultSet40((EmbedConnection)con, resultsToWrap, false, null, true);
        } catch (StandardException e) {
            throw new SQLException(Throwables.getRootCause(e));
        }
    }
}
```

We call the `getStatementStatistics` from custom procedure in our database, passing it an SQL query .
`getStatementStatistics` performs the following operations:

1. Query your database

The first step is to use our JDBC driver to connect to your database and run the query:

```
Connection con = DriverManager.getConnection("jdbc:default:connection");  
PreparedStatement ps = con.prepareStatement(statement);  
ResultSet rs = ps.executeQuery();
```

2. Convert the query results into a Spark RDD

Next, we convert the query's result set into a Spark RDD:

```
JavaRDD<LocatedRow> resultSetRDD = ResultSetToRDD(rs);
```

3. Calculate statistics

Next, we use Spark to collect statistics for the query, using private methods in our `SparkStatistics` class:

```
int[] fieldsToConvert = getFieldsToConvert(ps);  
MultivariateStatisticalSummary summary = getColumnStatisticsSummary(resultSetRDD, fieldsToConvert);
```

You can view the implementations of the `getFieldsToConvert` and `getColumnStatisticsSummary` methods in the [Appendix](#) at the end of this topic.

4. Return the results

Finally, we return the results:

```
IteratorNoPutResultSet resultsToWrap = wrapResults((EmbedConnection) con, getColumnStatistics(ps, summary, fieldsToConvert));  
resultSets[0] = new EmbedResultSet40((EmbedConnection)con, resultsToWrap, false, null, true);
```

Run a Sample Program to Use Our Class

Follow these steps to run a simple example program to use the Spark MLlib library to calculate statistics for an SQL statement.

1. Create Your API Class

The first step is to create a Java class that uses Spark to generate and analyze statistics, as shown in the previous section, [Creating our SparkStatistics Example Class](#)

2. Create your custom procedure

First we create a procedure in our database that references the `getStatementStatistics` method in our API, which takes an SQL query as its input and uses Spark to calculate statistics for the query using MLlib:

```
CREATE PROCEDURE getStatementStatistics(statement varchar(1024))
PARAMETER STYLE JAVA
LANGUAGE JAVA
READS SQL DATA
DYNAMIC RESULT SETS 1
EXTERNAL NAME 'com.splicemachine.example.SparkStatistics.getStatements
statistics';
```

3. Create a table to use

Let's create a very simple table to illustrate use of our procedure:

```
create table t( col1 int, col2 double);
insert into t values(1, 10);
insert into t values(2, 20);
insert into t values(3, 30);
insert into t values(4, 40);
```

4. Call your custom procedure to get statistics

Now call your custom procedure, which sends an SQL statement to the SparkStatistics class we created to generate a result set:

```
call splice.getStatementStatistics('select * from t');
```

Appendix: The SparkStatistics Class

Here's the full code for our SparkStatistics class:

```

package com.splicemachine.example;

import com.google.common.base.Throwables;
import com.google.common.collect.Lists;
import com.splicemachine.db.iapi.error.StandardException;
import com.splicemachine.db.iapi.sql.Activation;
import com.splicemachine.db.iapi.sql.ResultColumnDescriptor;
import com.splicemachine.db.iapi.sql.execute.ExecRow;
import com.splicemachine.db.iapi.types.DataTypeDescriptor;
import com.splicemachine.db.iapi.types.SQLDouble;
import com.splicemachine.db.iapi.types.SQLLongint;
import com.splicemachine.db.iapi.types.SQLVarchar;
import com.splicemachine.db.impl.jdbc.EmbedConnection;
import com.splicemachine.db.impl.jdbc.EmbedResultSet40;
import com.splicemachine.db.impl.sql.GenericColumnDescriptor;
import com.splicemachine.db.impl.sql.execute.IteratorNoPutResultSet;
import com.splicemachine.db.impl.sql.execute.ValueRow;
import com.splicemachine.derby.impl.sql.execute.operations.LocatedRow;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.mllib.linalg.Vector;
import org.apache.spark.mllib.stat.MultivariateStatisticalSummary;
import org.apache.spark.mllib.stat.Statistics;
import java.sql.*;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Types;
import java.util.List;

public class SparkStatistics {

    private static final ResultColumnDescriptor[] STATEMENT_STATS_OUTPUT_COLUMNS = new GenericColumnDescriptor[]{
        new GenericColumnDescriptor("COLUMN_NAME", DataTypeDescriptor.getBuiltInDataTypeDescriptor(Types.VARCHAR)),
        new GenericColumnDescriptor("MIN", DataTypeDescriptor.getBuiltInDataTypeDescriptor(Types.DOUBLE)),
        new GenericColumnDescriptor("MAX", DataTypeDescriptor.getBuiltInDataTypeDescriptor(Types.DOUBLE)),
        new GenericColumnDescriptor("NUM_NONZEROS", DataTypeDescriptor.getBuiltInDataTypeDescriptor(Types.DOUBLE)),
        new GenericColumnDescriptor("VARIANCE", DataTypeDescriptor.getBuiltInDataTypeDescriptor(Types.DOUBLE)),
        new GenericColumnDescriptor("MEAN", DataTypeDescriptor.getBuiltInDataTypeDescriptor(Types.DOUBLE)),
        new GenericColumnDescriptor("NORML1", DataTypeDescriptor.getBuiltInDataTypeDescriptor(Types.DOUBLE)),
        new GenericColumnDescriptor("MORML2", DataTypeDescriptor.getBuiltInDataTypeDescriptor(Types.DOUBLE)),
        new GenericColumnDescriptor("COUNT", DataTypeDescriptor.getBuiltInDataTypeDescriptor(Types.BIGINT)),
    };
}

```

```

public static void getStatementStatistics(String statement, ResultSet[] resultSetS
ts) throws SQLException {
    try {
        // Run sql statement
        Connection con = DriverManager.getConnection("jdbc:default:connection");
        PreparedStatement ps = con.prepareStatement(statement);
        ResultSet rs = ps.executeQuery();

        // Convert result set to Java RDD
        JavaRDD<JavaRDDLocatedRow> resultSetRDD = ResultSetToRDD(rs);

        // Collect column statistics
        int[] fieldsToConvert = getFieldsToConvert(ps);
        MultivariateStatisticalSummary summary = getColumnStatisticsSummary(resu
ltSetRDD, fieldsToConvert);

        IteratorNoPutResultSet resultsToWrap = wrapResults((EmbedConnection) co
n, getColumnStatistics(ps, summary, fieldsToConvert));
        resultSetS[0] = new EmbedResultSet40((EmbedConnection)con, resultsToWrap
, false, null, true);
    } catch (StandardException e) {
        throw new SQLException(Throwables.getRootCause(e));
    }
}

private static MultivariateStatisticalSummary getColumnStatisticsSummary(JavaRDD
LocatedRow> resultSetRDD,
                                         int[] fieldsToC
onvert) throws StandardException{
    JavaRDD<Vector> vectorJavaRDD = SparkMLibUtils.locatedRowRDDToVectorRDD(resul
tSetRDD, fieldsToConvert);
    MultivariateStatisticalSummary summary = Statistics.colStats(vectorJavaRDD.r
dd());
    return summary;
}

/*
 * Convert a ResultSet to JavaRDD
 */
private static JavaRDD<JavaRDDLocatedRow> ResultSetToRDD (ResultSet resultSet) throws St
andardException{
    EmbedResultSet40 ers = (EmbedResultSet40)resultSet;

    com.splicemachine.db.iapi.sql.ResultSet rs = ers.getUnderlyingResultSet();
    JavaRDD<JavaRDDLocatedRow> resultSetRDD = SparkMLibUtils.resultSetToRDD(rs);

    return resultSetRDD;
}

```

```

private static int[] getFieldsToConvert(PreparedStatement ps) throws SQLException {
    ResultSetMetaData metaData = ps.getMetaData();
    int columnCount = metaData.getColumnCount();
    int[] fieldsToConvert = new int[columnCount];
    for (int i = 0; i < columnCount; ++i) {
        fieldsToConvert[i] = i+1;
    }
    return fieldsToConvert;
}

/*
 * Convert column statistics to an iterable row source
 */
private static Iterable<ExecRow> getColumnStatistics(PreparedStatement ps,
                                                       MultivariateStatisticalSummary summary,
                                                       int[] fieldsToConvert) throws StandardException {
    try {

        List<ExecRow> rows = Lists.newArrayList();
        ResultSetMetaData metaData = ps.getMetaData();

        double[] min = summary.min().toArray();
        double[] max = summary.max().toArray();
        double[] mean = summary.mean().toArray();
        double[] nonZeros = summary.numNonzeros().toArray();
        double[] variance = summary.variance().toArray();
        double[] normL1 = summary.normL1().toArray();
        double[] normL2 = summary.normL2().toArray();
        long count = summary.count();

        for (int i= 0; i < fieldsToConvert.length; ++i) {
            int columnPosition = fieldsToConvert[i];
            String columnName = metaData.getColumnName(columnPosition);
            ExecRow row = new ValueRow(9);
            row.setColumn(1, new SQLVarchar(columnName));
            row.setColumn(2, new SQLDouble(min[columnPosition-1]));
            row.setColumn(3, new SQLDouble(max[columnPosition-1]));
            row.setColumn(4, new SQLDouble(nonZeros[columnPosition-1]));
            row.setColumn(5, new SQLDouble(variance[columnPosition-1]));
            row.setColumn(6, new SQLDouble(mean[columnPosition-1]));
            row.setColumn(7, new SQLDouble(normL1[columnPosition-1]));
            row.setColumn(8, new SQLDouble(normL2[columnPosition-1]));
            row.setColumn(9, new SQLLongint(count));
            rows.add(row);
        }
        return rows;
    }
}

```

```
        }
    catch (Exception e) {
        throw StandardException.newException(e.getLocalizedMessage());
    }
}

private static IteratorNoPutResultSet wrapResults(EmbedConnection conn, Iterable<ExecRow> rows) throws
    StandardException {
    Activation lastActivation = conn.getLanguageConnection().getLastActivatio
n();
    IteratorNoPutResultSet resultsToWrap = new IteratorNoPutResultSet(rows, STAT
EMENT_STATS_OUTPUT_COLUMNS,
        lastActivation);
    resultsToWrap.openCore();
    return resultsToWrap;
}
}
```

See Also

- » Spark Overview
- » Using the Splice Machine Database Console
- » You can find the Spark MLlib guide in the Programming Guides section of the Spark documentation site: <https://spark.apache.org/docs>

Using the Splice Machine Virtual Table Interface (VTI)

The Virtual Table Interface (VTI) allows you to use an SQL interface with data that is external to your database. This topic introduces the Splice Machine VTI in these sections:

- » [About VTI](#) describes the virtual table interface.
- » [Splice Machine Built-in Virtual Table Interfaces](#) describes the virtual table interfaces built into Splice Machine, and provides examples of using each.
- » [Creating a Custom Virtual Table Interface](#) walks you through the steps required to create a custom virtual table interface, and demonstrates how to simplify its use with a table function.
- » [The Splice Machine Built-in VTI Classes](#) provides reference descriptions of the virtual table interface classes built into by Splice Machine.

About VTI

You can use the Splice Machine Virtual Table Interface (*VTI*) to access data in external files, libraries, and databases.

NOTE: A virtual table is a view of data stored elsewhere; the data itself is not duplicated in your Splice Machine database.

The external data source can be any information source, including:

- » XML formatted reports and logs.
- » Queries that run in external databases that support JDBC, such as Oracle and DB2.
- » RSS feeds.
- » Flat files in formats such as comma-separated value (csv) format.

About Table Functions

A *table function* returns *ResultSet* values that you can query like you do tables that live in your Splice Machine database. A table function is bound to a constructor for a custom VTI class. Here's an example of a declaration for a table function that is bound to the `PropertiesFileVTI` class, which we walk you through implementing later in this topic:

```

CREATE FUNCTION propertiesFile(propertyFilename VARCHAR(200))
RETURNS TABLE
(
    KEY_NAME varchar(100)
    VALUE varchar(200)
)
LANGUAGE JAVA
PARAMETER STYLE SPLICE_JDBC_RESULT_SET
READS SQL DATA
EXTERNAL NAME 'com.splicemachine.tutorials.vti.PropertiesFileVTI.getPropertiesFileVTI';

```

Splice Machine Built-in Virtual Table Interfaces

Splice Machine provides two built-in VTI classes that you can use:

Class	Description	Implemented by
SpliceFileVTI	For querying delimited flat files, such as CSV files.	com.splicemachine.derby.vti.SpliceFileVTI
SpliceJDBCVTI	For querying data from external sources that support the JDBC API.	com.splicemachine.derby.vti.SpliceJDBCVTI

Each of these classes implements the `DatasetProvider` interface, which is used by Spark for creating execution trees, and the `VTCosting` interface, which is used by the Splice Machine optimizer.

SpliceFileVTI Example

For example, if we have an input file named `vtiInfile.csv` that contains this information:

```

sculligan,Relief Pitcher,27,08-27-2015,2015-08-27 08:08:08,06:08:08
jpeepers,Catcher,37,08-26-2015,2015-08-21 08:09:08,08:08:08
mbamburger,Manager,47,08-25-2015,2015-08-20 08:10:08,10:08:08
gbrown,Batting Coach,46,08-24-2015,2015-08-21 08:11:08,11:08:08
jardson,Left Fielder,34,08-23-2015,2015-08-22 08:12:08,11:08:08

```

We can use the `SpliceFileVTI` class to select and display the contents of our input file:

```

SELECT * FROM new com.splicemachine.derby.vti.SpliceFileVTI(
  '/<path>/data/vtiInfile.csv','','','') AS b
  (name VARCHAR(10), title VARCHAR(30), age INT, something VARCHAR(12), date_hired
TIMESTAMP, clock TIME);NAME          |TITLE           |AGE    |SOMETHING   |DATE_HIRE
D          |CLOCK
-----
---
sculligan |Relief Pitcher |27    |08X-27-2015 |2015-08-27 08:08:08.0 |06:08:08
jpeepers  |Catcher        |37    |08-26-2015  |2015-08-21 08:09:08.0 |08:08:08
mbamburger|Manager        |47    |08-25-2015  |2015-08-20 08:10:08.0 |10:08:08
gbrown    |Batting Coach  |46    |08-24-2015  |2015-08-21 08:11:08.0 |11:08:08
jardson   |Left Fielder   |34    |08-23X-2015 |2015-08-22 08:12:08.0 |11:08:08

5 rows selected

```

SpliceJDBCVTI Example

We can use the SpliceJDBCVTI class to select and display the contents a table in a JDBC-compliant database. For example, here we query a table stored in a MySQL database:

```

SELECT * FROM new com.splicemachine.derby.vti.SpliceJDBCVTI(
  'jdbc:mysql://localhost/hr?user=root&password=mysql-passwd', 'mySchema', 'myTable')
AS b
  (name VARCHAR(10), title VARCHAR(30), age INT, something VARCHAR(12), date_hired
TIMESTAMP, clock TIME);NAME          |TITLE           |AGE    |SOMETHING   |DATE_HIRE
D          |CLOCK
-----
---
sculligan |Relief Pitcher |27    |08X-27-2015 |2015-08-27 08:08:08.0 |06:08:08
jpeepers  |Catcher        |37    |08-26-2015  |2015-08-21 08:09:08.0 |08:08:08
mbamburger|Manager        |47    |08-25-2015  |2015-08-20 08:10:08.0 |10:08:08
gbrown    |Batting Coach  |46    |08-24-2015  |2015-08-21 08:11:08.0 |11:08:08
jardson   |Left Fielder   |34    |08-23X-2015 |2015-08-22 08:12:08.0 |11:08:08

5 rows selected

```

Creating a Custom Virtual Table Interface

You can create a custom virtual table interface by creating a class that implements the DatasetProvider and VTICosting interfaces, which are described below.

You can then use your custom VTI within SQL queries by using VTI syntax, as shown in the examples in the previous section. You can also create a table function for your custom VTI, and then call that function in your queries, which simplifies using your interface.

This section walks you through creating a custom virtual table interface that reads and displays the property keys and values in a properties file. This interface can be executed using a table function or by specifying its full method name in SQL statements.



The full code for this example is in the Splice [Community Sample Code Repository on Github](#).

The remainder of this section is divided into these subsections:

- » [Declare Your Class](#)
- » [Implement the Constructors](#)
- » [Implement Your Method to Generate Results](#)
- » [Implement Costing Methods](#)
- » [Implement Other DatasetProvider Methods](#)
- » [Use Your Custom Virtual Table Interface](#)

Declare Your Class

The first thing you need to do is to declare your public class; since we're creating an interface to read property files, we'll call our class `PropertiesFileVTI`. To create a custom VTI interface, you need to implement the following classes:

Class	Description
<code>DatasetProvider</code>	Used by Spark to construct execution trees.
<code>VTICosting</code>	Used by the Splice Machine optimizer to estimate the cost of operations.

Here's the declaration:

```
public class PropertiesFileVTI implements DatasetProvider, VTICosting {

    //Used for logging (and optional)
    private static final Logger LOG = Logger.getLogger(PropertiesFileVTI.class);

    //Instance variable that will store the name of the properties file that is being read
    private String fileName;

    //Provide external context which can be carried with the operation
    protected OperationContext operationContext;
```

Implement the Constructors

This section describes the constructors that we implement for our custom class:

- » You need to implement an empty constructor if you want to use your class in table functions:

```
public PropertiesFileVTI()
```

- » This is the signature used by invoking the VTI using the class name in SQL queries:

```
public PropertiesFileVTI(String pfileName)
```

- » This static constructor is called by the VTI - Table Function.

```
public static DatasetProvider getPropertiesFileVTI(String fileName)
```

Here's our implementation of the constructors for the `PropertiesFileVTI` class:

```
public PropertiesFileVTI() {}
public PropertiesFileVTI(String pfileName) {
    this.fileName = pfileName;
}

public static DatasetProvider getPropertiesFileVTI(String fileName) {
    return new PropertiesFileVTI(fileName);
}
```

Implement Your Method to Generate Results

The heart of your virtual table interface is the `DatasetProvider` method `getDataSet`, which you override to generate and return a `DataSet`. It's declaration looks like this:

```
DataSet<LocatedRow> getDataSet(
    SpliceOperation op,          // References the op at the top of the stack
    DataSetProcessor dsp,        // Mechanism for constructing the execution tree
    ExecRow execRow ) throws StandardException;
```

The `VTI` process calls this method to compute the `ResultSet` that it should return. Our `PropertiesFileVTI` implementation is shown here:

```

@Override
public DataSet<LocatedRow> getDataSet(SpliceOperation op, DataSetProcessor dsp, Exec
Row execRow) throws StandardException {
    operationContext = dsp.createOperationContext(op);

    //Create an arraylist to store the key-value pairs
    ArrayList<LocatedRow> items = new ArrayList<LocatedRow>();

    try {
        Properties properties = new Properties();

        //Load the properties file
        properties.load(getClass().getClassLoader().getResourceAsStream(fileName));

        //Loop through the properties and create an array
        for (String key : properties.stringPropertyNames()) {
            String value = properties.getProperty(key);
            ValueRow valueRow = new ValueRow(2);
            valueRow.setColumn(1, new SQLVarchar(key));
            valueRow.setColumn(2, new SQLVarchar(value));
            items.add(new LocatedRow(valueRow));
        }
    } catch (FileNotFoundException e) {
        LOG.error("File not found: " + this.fileName, e);
    } catch (IOException e) {
        LOG.error("Unexpected IO Exception: " + this.fileName, e);
    } finally {
        operationContext.popScope();
    }
    return new ControlDataSet<>(items);
}

```

Implement Costing Methods

The Splice Machine optimizer uses costing estimates to determine the optimal execution plan for each query. You need to implement several costing methods in your VTI class:

- » `getEstimatedCostPerInstantiation` returns the estimated cost to instantiate and iterate through your table function. Unless you have an accurate means of estimating this cost, simply return 0 in your implementation.

```

public double getEstimatedCostPerInstantiation( VTIEnvironment vtiEnv) throws S
QLEception;

```

- » `getEstimatedRowCount` returns the estimated row count for a single scan of your table function. Unless you have an accurate means of estimating this cost, simply return 0 in your implementation.

```

public double getEstimatedCostPerInstantiation( VTIEnvironment vtiEnv) throws S
QLEception;

```

- » `supportsMultipleInstantiations` returns a Boolean value indicating whether your table function's `ResultSet` can be instantiated multiple times in a single query. For our `PropertiesFileVTI` implementation of this method, we simply return `False`, since there's no reason for our function to be used that way.

```
public double supportsMultipleInstantiations( VTIEnvironment vtiEnv) throws SQLException;
```

NOTE: The VTICosting methods each take a `VTIEnvironment` argument; this is a state variable created by the Splice Machine optimizer, which methods can use to pass information to each other or to learn other details about the operating environment..

Here is the implementation of costing methods for our `PropertiesFileVTI` class:

```
@Override
public double getEstimatedCostPerInstantiation(VTIEnvironment arg0) throws SQLException {
    return 0;
}

@Override
public double getEstimatedRowCount(VTIEnvironment arg0) throws SQLException {
    return 0;
}

@Override
public boolean supportsMultipleInstantiations(VTIEnvironment arg0) throws SQLException {
    return false;
}
```

Implement Other DatasetProvider Methods

You also need to implement two additional `DatasetProvider` methods:

- » `getOperationContext` simply returns the current operation context (`this.operationContext`).

```
OperationContext getOperationContext();
```

- » `getMetaData` returns metadata that is used to dynamically bind your table function; this metadata includes a column descriptor for each column in your virtual table, including the name of the column, its type, and its size. In our `PropertiesFileVTI`, we assign the descriptors to a static variable, and our implementation of this method simply returns that value.

```
ResultSetMetaData getMetaData() throws SQLException;
```

Here is the implementation of these methods for our `PropertiesFileVTI` class:

```

@Override
public OperationContext getOperationContext() {
    return this.operationContext;
}

@Override
public ResultSetMetaData getMetaData() throws SQLException {
    return metadata;
}

private static final ResultColumnDescriptor[] columnInfo = {
    EmbedResultSetMetaData.getResultColumnDescriptor("KEY1", Types.VARCHAR, false, 200),
    EmbedResultSetMetaData.getResultColumnDescriptor("VALUE", Types.VARCHAR, false, 200),
};

private static final ResultSetMetaData metadata = new EmbedResultSetMetaData(columnInfo);
}

```

Use Your Custom Virtual Table Interface

You can create a table function in your Splice Machine database to simplify use of your custom VTI. Here's a table declaration for our custom interface:

```

CREATE FUNCTION propertiesFile(propertyFilename VARCHAR(200))
RETURNS TABLE
(
    KEY_NAME varchar(100)
    VALUE varchar(200)
)
LANGUAGE JAVA
PARAMETER STYLE SPLICE_JDBC_RESULT_SET
READS SQL DATA
EXTERNAL NAME 'com.splicemachine.tutorials.vti.PropertiesFileVTI.getPropertiesFileVTI';

```

You can now use your interface with table function syntax; for example:

```
select * from table (propertiesFile('sample.properties')) b;
```

You can also use your interface by using VTI syntax in an SQL query; for example:

```

select * from new com.splicemachine.tutorials.vti.PropertiesFileVTI('sample.properties')
as b (KEY_NAME VARCHAR(20), VALUE VARCHAR(100));

```

The Splice Machine Built-in VTI Classes

This section describes the built-in VTI classes:

- » [The SpliceFileVTI class](#)
- » [The SpliceJDBCVTI class](#)

The SpliceFileVTI Class

You can use the SpliceFileVTI class to apply SQL queries to a file, such as a csv file, as shown in the examples below.

Constructors

You can use the following constructor methods with the SpliceFileVTI class. Each creates a virtual table from a file:

```
public SpliceFileVTI(String fileName)

public SpliceFileVTI(String fileName,
                     String characterDelimiter,
                     String columnDelimiter)

public SpliceFileVTI(String fileName,
                     String characterDelimiter,
                     String columnDelimiter,
                     boolean oneLineRecords)
```

fileName

The name of the file that you are reading.

characterDelimiter

Specifies which character is used to delimit strings in the imported data. You can specify `null` or the empty string (' ') to use the default string delimiter, which is the double-quote (""). If your input contains control characters such as newline characters, make sure that those characters are embedded within delimited strings.

columnDelimiter

The character used to separate columns. Specify `null` if using the comma (,) character as your delimiter. Note that the backslash (\) character is not allowed as the column delimiter.

oneLineRecords

A Boolean value that specifies whether each line in the import file contains one complete record; if you specify `false`, records can span multiple lines in the input file.

The SpliceJDBCVTI Class

You can use the SpliceJDBCVTI class to access external databases that provide JDBC connections.

Constructors

You can use the following constructor methods with the SpliceJDBCVTI class.

```
public SpliceJDBCVTI(String connectionUrl,  
                      String schemaName,  
                      String tableName)
```

connectionURL

The URL of the database connection you are using.

schemaName

The name of the database schema.

tableName

The name of the table in the database schema.

```
public SpliceJDBCVTI(String connectionUrl,  
                      String sql)
```

connectionURL

The URL of the database connection you are using.

sql

The SQL string to execute in that database.

See Also

We recommend visiting the [Derby VTI documentation](#), which provides full reference documentation for the VTI class hierarchy.

Using the Splice Machine External Table Feature

This topic covers the use of external tables in Splice Machine. An external table references a file stored in a flat file format. You can use flat files that are stored in one of these formats:

- » ORC is a columnar storage format
- » PARQUET is a columnar storage format
- » Avro is a data serialization system
- » TEXTFILE is a plain text file

You can access ORC and PARQUET files that have been compressed with either Snappy or ZLIB compression; however, you cannot use a compressed plain text file.

About External Tables

You can use Splice Machine external tables to query the contents of flat files that are stored outside of your database. You query external tables pretty much the same way as you do the tables in your database.

External tables reference files that are stored in a flat file format such as Apache Parquet or Apache Orc, both of which are columnar storage formats that are available in Hadoop. You can use the `CREATE EXTERNAL TABLE` statement to create an external table that is connected to a specific flat file.

Using External Tables

This section presents information about importing data into an external table, and includes several examples of using external tables with Splice Machine.

Importing Data Into an External Table

You cannot import data directly into an external table; if you already have an external table in a compatible format, you can use `CREATE EXTERNAL TABLE` statement to point at the external file and query against it.

If you want to create an external file from within Splice Machine, follow these steps:

1. Create (or use) a table in your Splice Machine database (your internal table).
2. Use `CREATE EXTERNAL TABLE` to create your empty external table, specifying the location where you want that data stored externally.
3. Use `INSERT INTO` (your external table) `SELECT` (from your internal table) to populate the external file with your data.
4. You can now query the external table.

Accessing a Parquet File

The following statement creates an external table for querying a PARQUET file that is stored on your computer:

```
splice> CREATE EXTERNAL TABLE myExtTbl (
    col1 INT, col2 VARCHAR(24))
    PARTITIONED BY (col1)
    STORED AS PARQUET
    LOCATION '/users/myname/myParquetFile';0 rows inserted/updated/deleted
```

The call to `CREATE EXTERNAL TABLE` associates a Splice Machine external table with the file named `myParquetFile`, and tells Splice Machine that:

- » The table should be partitioned based on the values in `col1`.
- » The file is stored in PARQUET format.
- » The file is located in `/users/myname/myParquetFile`.

After you create the external table, you can query `myExtTbl` just as you would any other table in your database.

Accessing and Updating an ORC File

The following statement creates an external table for creates an external table for an ORC file and inserts data into it:

```
splice> CREATE EXTERNAL TABLE myExtTbl2
    (col1 INT, col2 VARCHAR(24))
    PARTITIONED BY (col1)
    STORED AS ORC
    LOCATION '/users/myname/myOrcFile';
0 rows inserted/updated/deleted
splice> INSERT INTO myExtTbl2 VALUES (1, 'One'), (2, 'Two'), (3, 'Three');
3 rows inserted/updated/deleted
splice> SELECT * FROM myExtTbl2;
COL1      | COL2
-----
3        | Three
2        | Two
1        | One
```

The call to `CREATE EXTERNAL TABLE` associates a Splice Machine external table with the file named `myOrcFile`, and tells Splice Machine that:

- » The table should be partitioned based on the values in `col1`.
- » The file is stored in ORC format.
- » The file is located in `/users/myname/myOrcFile`.

The call to `INSERT INTO` demonstrates that you can insert values into the external table just as you would with an ordinary table.

Accessing a Plain Text File

You can specify a table constraint on an external table; for example:

```
splice> CREATE EXTERNAL TABLE myTextTable(
    col1 INT, col2 VARCHAR(24))
PARTITIONED BY (col1)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ESCAPED BY '\\'
LINES TERMINATED BY '\\n'
STORED AS TEXTFILE
LOCATION '/users/myName/myTextFile';0 rows inserted/updated/deleted
```

The call to `CREATE EXTERNAL TABLE` associates a Splice Machine external table with the file named `myOrcFile`, and tells Splice Machine that:

- » The table should be partitioned based on the values in `col1`.
- » Each field in each row is terminated by a comma.
- » Each line in the file is terminated by a line-end character.
- » The file is stored in plain text format.
- » The file is located in `/users/myName/myTextFile`.

Accessing a Compressed File

This example is exactly the same as our first example, except that the source file has been compressed with Snappy compression:

```
splice> CREATE EXTERNAL TABLE myExtTbl (
    col1 INT, col2 VARCHAR(24))
COMPRESSED WITH SNAPPY
PARTITIONED BY (col1)
STORED AS PARQUET
LOCATION '/users/myname/myParquetFile';
0 rows inserted/updated/deleted
```

Manually Refreshing an External Tables

If the schema of the file represented by an external table is updated, Splice Machine needs to refresh its representation. When you use the external table, Spark caches its schema in memory to improve performance; as long as you are using Spark to modify the table, it is smart enough to refresh the cached schema. However, if the table schema is modified outside of Spark, you need to call the `SYSCS_UTIL.SYSCS_REFRESH_EXTERNAL_TABLE` built-in system procedure. For example:

```
splice> CALL SYSCS_UTIL.SYSCS_REFRESH_EXTERNAL_TABLE('APP', 'myExtTable');
Statement executed.
```

See Also

The `CREATE EXTERNAL TABLE` statement.

The `SYSICS_UTIL.SYSCS_REFRESH_EXTERNAL_TABLE` built-in system procedure.

Using Splice Machine with HCatalog

This is an On-Premise-Only topic! [Learn about our products](#)

Apache HCatalog is a metadata and table management system for the broader Hadoop platform. HCatalogs table abstraction presents users with a relational view of data in the Hadoop distributed file system (HDFS) and ensures that users need not worry about where or in what format their data is stored. HCatalog supports reading and writing files in any format for which a SerDe (serializer-deserializer) can be written.

Splice Machine integrates with HCatalog, allowing any HiveQL statements to read from (e.g. SELECT) and write to (e.g. INSERT) Splice Machine tables. You can also use joins and unions to combine native Hive tables with Splice Machine tables.

You can find extensive information about HCatalog on the [Apache Hive web site](#).

Also note that you can also take advantage of our HCatalog integration to access your Splice Machine tables for reading and writing from any database that features HCatalog integration, such as *MongoDB*.

Using HCatalog with a Splice Machine Table

To use HCatalog with Splice Machine, you connect a Hive table with a Splice Machine table using the HiveQL CREATE EXTERNAL TABLE statement. In Hive, an external table can point to any HDFS location for its storage; in this case, the table is located in your Splice Machine database.

For example, here's a HiveQL statement that creates a table named extTest1 that is connected with the hcattest table in a Splice Machine database:

```
hive> CREATE EXTERNAL TABLE extTest1(col1 int, col2 varchar(20))
      STORED BY 'com.splicemachine.mrio.api.hive.SMStorageHandler'
      TBLPROPERTIES ("splice.jdbc"="jdbc:splice://localhost:1527/splicedb\;user=splic
e\;password=admin", "splice.tableName"="TEST.hcattest");
```

Your table definition must include:

- » The STORED BY clause as shown, which allows the table to connect with Splice Machine.
- » The TBLPROPERTIES clause, which specifies:
 - » The JDBC connection you are using. include your access **user** ID and **password**.

If you are running Splice Machine on a cluster, connect from a machine that is NOT running an HBase RegionServer and specify the IP address of a **regionServer** node, e.g. 10.1.1.110.

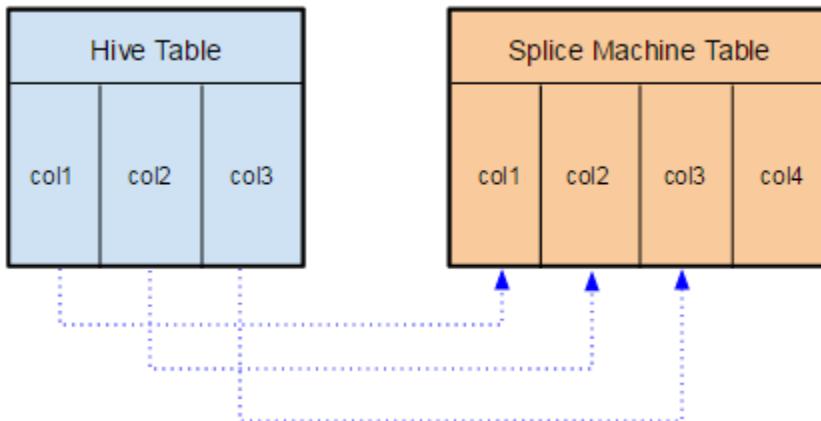
NOTE: Use **localhost** if you're running the standalone version of Splice Machine.

- » The name of the table in your Splice Machine database with which you are connecting the Hive table.

How Splice Machine Maps Columns

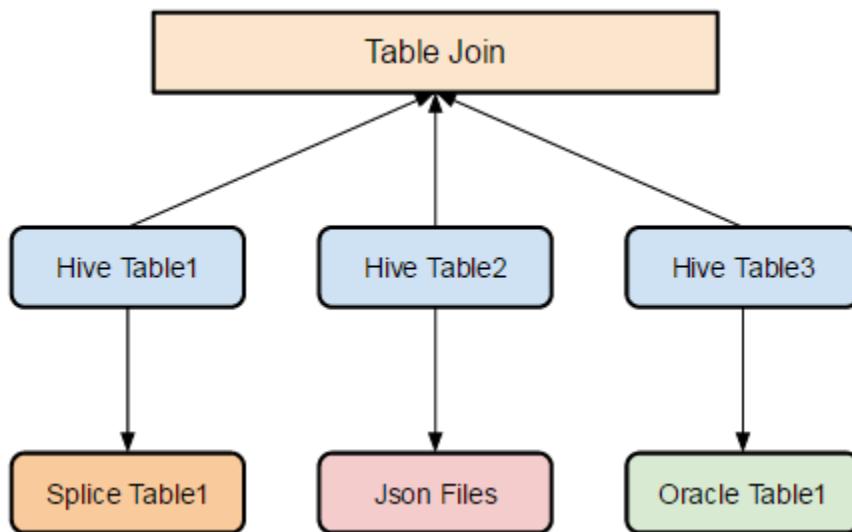
If your Hive table contains a different number of columns than does your Splice Machine database table, Splice Machine maps the columns.

For example, if your Hive table has three columns defined, and your Splice Machine table has four columns defined, then Splice Machine maps the three Hive columns into the first three columns in the Splice Machine table, as shown here:



Using HiveQL to Join Tables From Different Data Resources

You can use HiveQL to join tables that point to different data resources. The Java class provided by Splice Machine allows you to include tables from your Splice Machine database in such joins, as illustrated here:



For more information about Hive joins, see the Apache Hive documentation.

Examples

This section contains several simple examples of using HCatalog.

Example 1: Simple Example

This is a simple example of creating an external table in Hive that maps directly to a Splice Machine table.

1. Create the Splice Machine table

This example uses a very simple Splice Machine database table, hcattest, which we create with these statements:

```
splice> create table hcattest(col1 int, col2 varchar(20));
splice> insert into hcattest values(1, 'row1');
splice> insert into hcattest values(2, 'row2');
splice> insert into hcattest values(3, 'row3');
splice> insert into hcattest values(4, 'row4');
```

2. Verify the table

Verify that hcattest is set up correctly:

```
splice> describe hcattest;
COLUMN_NAME | TYPE_NAME | DES&| NUM&| COLUMN&| COLUMN_DEF| CHAR_OCTE&| IS_NUL
L&
-----
-
COL1        | INTEGER      | 0      | 10    | 10          | NULL       | NULL      | YES
COL2        | VARCHAR       | NULL   | NULL  | 20          | NULL       | 40        | YES

2 rows selected
splice> select * from hcattest;
COL1      |COL2
-----
1        |row1
2        |row2
3        |row3
4        |row4

4 rows selected
```

3. Create an external table in Hive

You need to create an external table in Hive, and connect that table with the Splice Machine table you just created. Type the following command into the Hive shell, substituting your `user` ID and `password`.

```
hive> CREATE EXTERNAL TABLE extTest1(col1 int, col2 varchar(20))
      STORED BY 'com.splicemachine.mrio.api.hive.SMStorageHandler'
      TBLPROPERTIES ("splice.jdbc"="jdbc:splice://localhost:1527/splicedb\;user=splice\;password=admin", "splice.tableName"="SPLICE.hcattest");
```

4. Use HiveQL to select data in the Splice Machine table

Once you've created your external table, you can use `SELECT` statements in the Hive shell to retrieve data from Splice Machine table. For example:

```
hive> select * from extTest1;
OK
1row 1
2row 2
3row 3
4row 4
```

If the external table that you created does not have the same number of columns as are in your Splice Machine table, then the Hive table's columns are mapped to columns in the Splice Machine table, as described in [How Splice Machine Maps Columns](#), above.

Example 2: Table with Primary Key

This example uses a Splice Machine table, `tblA`, that has only string types and a primary key.

1. Create and verify the Splice Machine table

This example uses a simple Splice Machine database table that we've created, `tblA`, which we verify:

```
splice> describe tblA;
COLUMN_NAME | TYPE_NAME | DES&| NUM&| COLUMN&| COLUMN_DEF| CHAR_OCTE&| IS_NUL
L&
-----
-
COL1        | CHAR        | NULL | NULL| 20        | NULL        | 40          | YES
COL2        | VARCHAR     | NULL | NULL| 56        | NULL        | 112         | YES

2 rows selected
splice> select * from tblA;
COL1      |COL2
-----
char      |varchar 2
char 1    |varchar 1

2 rows selected
```

2. Create an external table in Hive

You need to create an external table in Hive, and connect that table with the Splice Machine table you just created. Type the following command into the Hive shell, substituting your `user` ID and `password`.

```
hive> CREATE EXTERNAL TABLE extTest2(col1 String, col2 varchar(56))
      STORED BY 'com.splicemachine.mrio.api.hive.SMStorageHandler'
      TBLPROPERTIES ("splice.jdbc"="jdbc:splice://localhost:1527/splicedb\;user=splice\;password=admin", "splice.tableName"="SPICE.tblA");
```

3. Use HiveQL to select data in the Splice Machine table

Once you've created your external table, you can use SELECT statements in the Hive shell to retrieve data from Splice Machine table. For example:

```
hive> select * from extTest2;
OK
char  varchar 2
char  1varchar 1
```

Example 3: Table with No Primary Key

This example uses a Splice Machine table, `tblB`, that has a primary key and uses integer columns.

1. Create and verify the Splice Machine table

This example uses a simple Splice Machine database table that we've created, `tblA`, which we verify:

```
splice> describe tblA;
COLUMN_NAME | TYPE_NAME | DES&| NUM&| COLUMN&| COLUMN_DEF | CHAR_OCTE& | IS_NUL
L&
-----
-
COL1        | INTEGER    | 0      | 10   | 10       | NULL      | NULL      | NO
COL2        | INTEGER    | 0      | 10   | 10       | NULL      | NULL      | YES
COL3        | INTEGER    | 0      | 10   | 10       | NULL      | NULL      | NO

3 rows selected
splice> select * from tblB;
COL1      | COL2      | COL3
-----
1          | 1          | 1
2          | 2          | 2

2 rows selected
```

2. Create an external table in Hive

You need to create an external table in Hive, and connect that table with the Splice Machine table you just created. Type the following command into the Hive shell, substituting your `user` ID and `password`.

```
hive> CREATE EXTERNAL TABLE extTest3(col1 String, col2 varchar(56))
      STORED BY 'com.splicemachine.mrio.api.hive.SMStorageHandler'
      TBLPROPERTIES ("splice.jdbc"="jdbc:splice://localhost:1527/splicedb\;user=splice\;password=admin", "splice.tableName"="SPLICE.tblB");
```

3. Select data from the Splice Machine input table

Once you've created your external table, you can use SELECT statements in the Hive shell to retrieve data from Splice Machine input table. For example:

```
hive> select * from extTest3;
OK
111
122
```

Configuring Load Balancing and High Availability with HAProxy

HAProxy is an open source utility that is available on most Linux distributions and cloud platforms for load-balancing TCP and HTTP requests. Users can leverage this tool to distribute incoming client requests among the region server nodes on which Splice Machine instances are running.

The advantages of using HAProxy with Splice Machine clusters are:

- » Users need to point to only one JDBC host and port for one Splice Machine cluster, which may have 100s of nodes.
- » The HAProxy service should ideally be running on a separate node that is directing the traffic to the region server nodes; this means that if one of the region server node goes down, users can still access the data from another region server node.
- » The load balance mechanism in HAProxy helps distribute the workload evenly among the set of nodes; you can optionally select this algorithm in your configuration, which can help increase throughput rate.

The remainder of this topic walks you through configuring HAProxy on a non-Splice Machine node that is running Red Hat Enterprise Linux.



Our Tutorials chapter includes a topic that addresses Using HAProxy on a Kerberos-enabled Cluster.

Configuring HAProxy with Splice Machine

The following example shows you how to configure HAProxy load balancer on a non-Splice Machine node on a Red Hat Enterprise Linux system. Follow these steps:

1. Install HAProxy as superuser :

```
# yum install haproxy
```

2. Configure the /etc/haproxy/haproxy.cfg file, following the comments in the sample file below:

In this example, we set the incoming requests to `haproxy_host:1527`, which uses a balancing algorithm of least connections to distribute among the nodes `srv127`, `srv128`, `srv129`, and `srv130`. This means that the incoming connection is routed to the region server that has the least number of connections; thus, the client JDBC URL should point to `<haproxy_host>:1527`.

NOTE: The HAProxy manual describes other balancing algorithms that you can use.

Here is the `haproxy.cfg` file for this example:

```

#-----
# Global settings
#-----
global
    # to have these messages end up in /var/log/haproxy.log you will
    # need to:
    #
    # 1) configure syslog to accept network log events. This is done
    #     by adding the '-r' option to the SYSLOGD_OPTIONS in
    #     /etc/sysconfig/syslog
    #
    # 2) configure local2 events to go to the /var/log/haproxy.log
    #     file. A line like the following can be added to
    #     /etc/sysconfig/syslog
    #
    #     local2.*                  /var/log/haproxy.log
    #
maxconn 4000
log 127.0.0.1 local2
user haproxy
group haproxy

#-----
# common defaults that all the 'listen' and 'backend' sections will
# use if not designated in their block
#-----
defaults
    log global
    retries 2
    timeout connect 30000
    timeout server 50000
    timeout client 50000

#-----
# This enables jdbc/odbc applications to connect to HAProxy_host:1527 por
t
# so that HAProxy can balance between the splice engine cluster nodes
# where each node's splice engine instance is listening on port 1527
#-----
listen splice-cluster
    bind *:1527
    log global
    mode tcp
    option tcplog
    option tcp-check
    option log-health-checks
    timeout client 3600s

```

```

timeout server 3600s
balance leastconn
server srv127 10.1.1.227:1527 check
server srv128 10.1.1.228:1527 check
server srv129 10.1.1.229:1527 check
server srv130 10.1.1.230:1527 check

#-----
# (Optional) set up the stats admin page at port 1936
#-----
listen stats :1936
    mode http
    stats enable
    stats hide-version
    stats show-node
    stats auth admin:password
    stats uri /haproxy?stats

```

Note that some of the parameters may need tuning per the sizing and workload nature:

- » The `maxconnections` parameter indicates how many concurrent connections are served at any given time; you may need to configure this, based on size of the cluster and expected inbound requests.
- » Similarly, the `timeout` values, which are by default in msec, should be tuned so that the connection does not get terminated while a long-running query is executed.

3. Start the HAProxy service:

As superuser, follow these steps to enable the HAProxy service:

Distribution	Instructions
Redhat / CentOS EL6	<pre># chkconfig haproxy on # service haproxy start</pre> <p>If you change the configuration file, reload it with this command:</p> <pre># service haproxy reload</pre>

Distribution	Instructions
Redhat / CentOS EL7	<pre># systemctl enable haproxy ln -s '/usr/lib/systemd/system/haproxy.service' '/etc/systemd/system/multi-user.target.wants/haproxy.service' # systemctl start haproxy</pre> <p>If you change the configuration file, reload it with this command:</p> <pre># systemctl haproxy reload</pre>

NOTE: You can find the HAProxy process id in: `/var/run/haproxy.pid`. If you encounter any issues starting the service, check if SELinux is enabled; you may want to disable it initially.

4. Connect:

You can now connect JDBC clients, including the Splice Machine command line interpreter, `sqlshell.sh`. Use the following JDBC URL:

```
jdbc:splice://<haproxy_host>:1527/splicedb;user=splice;password=admin
```

For ODBC clients to connect through HAProxy, ensure that the DSN entry in file `.odbc.ini` is pointing to the HAProxy host.

5. Verify that inbound requests are being routed correctly:

You can check the logs at `/var/log/haproxy.log` to make sure that inbound requests are being routed to Splice Machine region servers that are receiving inbound requests on port 1527.

6. View traffic statistics:

If you have enabled HAProxy stats, as in our example, you can view the overall traffic statistics in browser at:

```
http://<haproxy_host>:1936/haproxy?stats
```

You'll see a report that looks similar to this:

HAProxy

Statistics Report for pid 751 on stl-col0-10-1-0-236.splicemachine.col0

> General process information

pid = 751 (process #1, nbproc = 1)
 uptime = 0d 0h37m00s
 system limits: memmax = unlimited; ulimit-n = 4016
 maxsock = 4016; maxconn = 2000; maxpipes = 0
 current conn = 1001; current pipes = 0/0; conn rate = 1/sec
 Running tasks: 1/1008; idle = 96 %

active UP
 active UP, going down
 active DOWN, going up
 active or backup DOWN
 active or backup DOWN for maintenance (MAINT)
 active or backup SOFT STOPPED for maintenance
 Note: "NOLB"?"DRAIN" = UP with load-balancing disabled.

- Display option:
- + Scope :
- Hide "DOWN" #
- Disable refresh
- Refresh now
- CSV export

splice-cluster																											
	Queue			Session rate			Sessions						Bytes			Denied			Errors			Warnings			Status		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act		
Frontend	0	0	-	0	116	-	1 000	1 000	2 000	3 001			1 244 674	70 579 321	0	0	0					OPEN					
svr127	0	0	-	0	29	-	250	250	-	751	751	5m14s	473 674	69 991 321	0		0	0	0	0	0	0	37m UP	L4OK in 0ms	1	Y	
svr128	0	0	-	0	29	-	250	250	-	750	750	5m14s	257 000	198 000	0		0	0	0	0	0	0	0	37m UP	L4OK in 0ms	1	Y
svr129	0	0	-	0	29	-	250	250	-	750	750	5m14s	257 000	198 000	0		0	0	0	0	0	0	0	37m UP	L4OK in 0ms	1	Y
svr130	0	0	-	0	29	-	250	250	-	750	750	5m14s	257 000	198 000	0		0	0	0	0	0	0	0	37m UP	L4OK in 0ms	1	Y
Backend	0	0	-	0	116	-	1 000	1 000	2 000	3 001	3 001	5m14s	1 244 674	70 579 321	0	0	0	0	0	0	0	0	0	37m UP		4	4
state																											
	Queue			Session rate			Sessions						Bytes			Denied			Errors			Warnings			Status		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act		
Frontend	1	4	-	1	3	-	2 000	85					61 755	1 118 907	0	0	12						OPEN				
Backend	0	0	-	0	2	-	0	1	200	71	0	0s	61 755	1 118 907	0	0	0	71	0	0	0	0	37m UP		0	0	

Splice Machine Map Reduce API

This is an On-Premise-Only topic! [Learn about our products](#)

The Splice Machine MapReduce API provides a simple programming interface to the Map Reduce Framework that is integrated into Splice Machine. You can use MapReduce to import data, export data, or for purposes such as implementing machine learning algorithms. One likely scenario for using the Splice Machine MapReduce API is for customers who already have a Hadoop cluster, want to use Splice Machine as their transactional database, and need to continue using their batch MapReduce jobs.

This topic includes a summary of the Java classes included in the API, and [presents an example](#) of using the MapReduce API.

Splice Machine MapReduce API Classes

The Splice Machine MapReduce API includes the following key classes:

Class	Description
SpliceJob	Creates a transaction for the MapReduce job.
SMInputFormat	Creates an object that: <ul style="list-style-type: none"> » uses Splice Machine to scan the table and decode the data » returns an ExecRow (typed data) object
SMOoutputFormat	Creates an object that: <ul style="list-style-type: none"> » writes to a buffered cache » dumps the cache into Splice Machine » returns an ExecRow (typed data) object.
SpliceMapReduceUtil	A Helper class for writing MapReduce jobs in java; this class is used to initiate a mapper job or a reducer job, to set the number of reducers, and to add dependency jars.

NOTE: Each transaction must manage its own commit and rollback operations.

For information about and examples of using Splice Machine with HCatalog, see the [Using Splice Machine with HCatalog](#) topic.

Example of Using the Splice Machine MapReduce API

This topic describes using the Splice Machine MapReduce API, `com.splicemachine.mrio.api`, a simple word count program that retrieves data from an input table, summarizes the count of initial character of each word, and writes the result to an output table.

1. Define your input and output tables:

First, assign the name of the Splice Machine database table from which you want to retrieve data to a variable, and then assign a name for your output table to another variable:

```
String inputTableName = "WIKIDATA";
String outputTableName = "USERTEST";
```

You can specify table names using the `<schemaName>.<tableName>` format; if you don't specify a schema name, the default schema is assumed.

2. Create a new job instance:

You need to create a new job instance and assign a name to it:

```
Configuration config = HBaseConfiguration.create();
Job job = new Job(config, "WordCount");
```

3. Initialize your mapper job:

We initialize our sample job using the `initTableMapperJob` utility method:

```
TableMapReduceUtil.initTableMapperJob(
    tableName, // input Splice Machine database table
    scan, // a scan instance to control CF and attribute selection
    MyMapper.class, // the mapper
    Text.class, // the mapper output key
    InitWritable.class, // the mapper output value
    job,
    true,
    SpliceInputFormat.class);
```

4. Retrieve values within your map function:

Our sample `map` function retrieves and parses a single row with specified columns.

```

public void map(ImmutableBytesWritable row, ExecRow value, Context context)
                throws InterruptedException, IOException {
    if(value != null) {
        try {
            DataValueDescriptor dataValDesc[] = value.getRowArray();
            if(dataValDesc[0] != null) {}
            word = dataValDesc[0].getString();
        }
        catch (StandardException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        if(word != null) {
            Text key = new Text(word.charAt(0)+"");
            IntWritable val = new IntWritable(1);
            context.write(key, val);
        }
    }
}

```

5. Manipulate and save the value with reduce function:

Our sample reduce function manipulates and saves the value by creating an ExecRow and filling in the row with the execRow.setRowArray method.

```

public void reduce(Text key, IterableIntWritable> values, Context context)
                throws IOException, InterruptedException {

    IteratorIntWritable> it=values.iterator();
    ExecRow execRow = new ValueRow(2);
    int sum = 0;
    String word = key.toString();
    while (it.hasNext()) {
        sum += it.next().get();
    }
    try{
        DataValueDescriptor []dataValDescs= {new SQLVarchar(word), new SQLInteger(sum)};
        execRow.setRowArray(dataValDescs);
        context.write(new ImmutableBytesWritable(Bytes.toBytes(word)), execRow);
    }
    catch(Exception E) {
        E.printStackTrace();
    }
}

```

6. Commit or rollback the job:

If the job is successful, commit the transaction.

```
job.commit();
```

If the job fails, roll back the transaction.

```
job.rollback();
```

Working with HBase

This is an On-Premise-Only topic! [Learn about our products](#)

This topic presents information about working in HBase with Splice Machine, in these sections:

- » The [Mapping Splice Machine Tables to HBase](#) section shows you how to view your Splice Machine tables and how map the table names that we see in HBase to the more descriptive table names that we see in Splice Machine.
- » The [Accessing HBase Master](#) section shows you how to access HBase Master so that you can peek under the surface a bit.

Mapping Splice Machine Tables to HBase

If you are looking at tables in the Splice Machine database that do not appear to match what you see in the HBase Master Web user interface, they may not match up. To view your Splice Machine tables in HBase, follow these steps:

1. Use the Splice interactive command interface to view the tables:

You can use the `show tables` command to view the tables in your Splice Machine database:

TABLE_SCHEM	TABLE_NAME	CONGLOM_ID	REMARKS
SYS	SYSALIASES	368	
SYS	SYSBACKUP	944	
SYS	SYSBACKUPFILESET	976	
SYS	SYSBACKUPITEMS	1056	
SYS	SYSBACKUPJOBS	1184	
SYS	SYSCHECKS	384	
SYS	SYSCOLPERMS	640	
SYS	SYSCOLUMNS	80	
SYS	SYSCOLUMNSTATS	1216	
SYS	SYSCONGLOMERATES	48	
SYS	SYSCONSTRAINTS	336	
SYS	SYSDEPENDS	272	
SYS	SYSFILES	288	
SYS	SYSFOREIGNKEYS	240	
SYS	SYSKEYS	320	
SYS	SYSPERMS	816	
SYS	SYSPHYSICALSTATS	1264	
SYS	SYSPRIMARYKEYS	1424	
SYS	SYSROLES	752	
SYS	SYSROUTINEPERMS	672	
SYS	SYSSCHEMAS	32	
SYS	SYSSEQUENCES	800	
SYS	SYSSTATEMENTS	256	
SYS	SYSTABLEPERMS	656	
SYS	SYSTABLES	64	
SYS	SYSTABLESTATS	1280	
SYS	SYSTRIGGERS	304	
SYS	SYSUSERS	880	
SYS	SYSVIEWS	352	
SYSIBM	SYSUMMY1	1296	

2. View the tables in the HBase Master Web Console:

To view the HBase tables, use the HBase Master Web Console, at <http://localhost:60010/>.

Note that all of the user tables in the *Tables* section have numeric names; the numbers match the conglomerate number (CONGLOM_ID) values displayed by the `show tables` command.

Tables

User Tables

System Tables

Snapshots

93 table(s) in set. [Details]

Namespace	Table Name	Online Regions	Description
splice	1009	1	'splice:1009', {TABLE_ATTRIBUTES => {METADATA => {'indexDisplayName' => 'SYSCOLPERMS_INDEX3', 'tableDisplayName' => 'SYSCOLPERMS'}}, {NAME => 'V', VERSIONS => '2147483647'}}
splice	1025	1	'splice:1025', {TABLE_ATTRIBUTES => {METADATA => {'indexDisplayName' => 'SYSROLES_INDEX2', 'tableDisplayName' => 'SYSROLES'}}, {NAME => 'V', VERSIONS => '2147483647'}}
splice	1041	1	'splice:1041', {TABLE_ATTRIBUTES => {METADATA => {'indexDisplayName' => 'SYSUSERS_INDEX1', 'tableDisplayName' => 'SYSUSERS'}}, {NAME => 'V', VERSIONS => '2147483647'}}

These numbers are used in HBase as directory names; you can find those directories in your file system and examine the tables directly.

Accessing HBase Master

If you are an HBase veteran or someone interested in seeing a little of what goes on under the surface, you can access HBase Master with the default HBase URL:

`http://localhost:60010`

Because Splice Machine encodes and compresses the data for space efficiency, the actual data in your tables is virtually unreadable.

NOTE: In this version, Splice Machine has purposely left ports 60010 and 60030 open for people to see the HBase tables. If security is an issue in your deployment, you can easily block this access.

Using Functions and Stored Procedures

This topic provides an overview of writing and using functions and stored procedures in Splice Machine.

About User-Defined Functions

You can create user-defined database functions that can be evaluated in SQL statements; these functions can be invoked where most other built-in functions are allowed, including within SQL expressions and `SELECT` statement. Functions must be deterministic, and cannot be used to make changes to the database.

You can create two kinds of functions:

- » Scalar functions, which always return a single value (or `NULL`),
- » Table functions, which return a table.

When you invoke a function within a `SELECT` statement, it is applied to each retrieved row. For example:

```
SELECT ID, Salary, MyAdjustSalaryFcn(Salary) FROM SPLICEBBALL.Salaries;
```

This `SELECT` will execute the `MyAdjustSalaryFcn` to the `Salary` value for each player in the table.

About Stored Procedures

You can group a set of SQL commands together with variable and logic into a stored procedure, which is a subroutine that is stored in your database's data dictionary. Unlike user-defined functions, a stored procedure is not an expression and can only be invoked using the `CALL` statement. Stored procedures allow you to modify the database and return Result Sets or nothing at all.

Stored procedures can be used for situations where a complex set of SQL statements are required to process something, and that process is used by various applications; creating a stored procedure increases performance efficiency. They are typically used for:

- » checking business rules and validating data before performing actions
- » performing significant processing of data with the inputs to the procedure

Comparison of Functions and Stored Procedures

Here's a comparison of Splice Machine functions and stored procedures:

Database Function	Stored Procedure
<p>A Splice Machine database function:</p> <ul style="list-style-type: none"> » must be written as a public static method in a Java public class » is executed in exactly the same manner as are public static methods in Java » can have multiple input parameters » always returns a single value (which can be null) » cannot modify data in the database 	<p>Splice Machine stored procedures can:</p> <ul style="list-style-type: none"> » return result sets or return nothing at all » issue update, insert, and delete statements » perform DDL statements such as create and drop » consolidate and centralize code » reduce network traffic and increase execution speed
<p>Can be used in SELECT statements.</p>	<p>Cannot be used in SELECT statements. Must be invoked using a CALL statement.</p>
<p>Create with the CREATE FUNCTION statement, which is described in our SQL Reference book. You can find an example in the Function and Stored Procedure Examples topic in this section.</p>	<p>Create with the CREATE PROCEDURE statement, which is described in our SQL Reference book. You can find an example in the Function and Stored Procedure Examples topic in this section.</p>

Operations in Which You Can Use Functions and Stored Procedures

The following table provides a list of the differences between functions and stored procedures with regard to when and where they can be used:

Operation	Functions	Stored Procedures
<i>Execute in an SQL Statement</i>	Yes	No
<i>Execute in a Trigger</i>	Yes	Triggers that execute before an operation (<i>before triggers</i>) cannot modify SQL data.
<i>Process OUT / INOUT Parameters</i>	No	Yes
<i>Return Resultset(s)</i>	No	Yes
<i>Execute SQL Select</i>	Yes	Yes

Operation	Functions	Stored Procedures
<i>Execute SQL Update/ Insert/Delete</i>	No	Yes
<i>Execute DDL (Create/Drop)</i>	No	Yes

Viewing Functions and Stored Procedures

You can use the `show functions` and `show procedures` commands in the `splice>` command line interface to display the functions and stored procedures available in your database:

Command	Output
<code>splice> show functions;</code>	All functions defined in your database
<code>splice> show functions in SYSCS_UTIL;</code>	All functions in the <code>SYSCS_UTIL</code> schema in your database
<code>splice> show procedures;</code>	All stored procedures defined in your database
<code>splice> show procedures in SYSCS_UTIL;</code>	All stored procedures in the <code>SYSCS_UTIL</code> schema in your database

Writing and Deploying Functions and Stored Procedures

The remainder of this section presents information about and examples of writing functions and stored procedures for use with Splice Machine, in these topics:

- » [Writing Functions and Stored Procedures](#) shows you the steps required to write functions stored procedures and add them to your Splice Machine database.
- » [Storing and Updating Functions and Stored Procedures](#) tells you how to store new JAR files, replace JAR files, and remove JAR files in your Splice Machine database.
- » [Examples of Splice Machine Functions and Stored Procedures](#) provides you with examples of functions and stored procedures.

See Also

- » [CREATE FUNCTION](#)
- » [CREATE PROCEDURE](#)

Writing Functions and Stored Procedures

This topic shows you the steps required to write functions and stored procedures for use in your Splice Machine database.

- » Refer to the [Introduction to Functions and Stored Procedures](#) topic in this section for an overview and comparison of functions and stored procedures.
- » Refer to the [Storing and Updating Functions and Stored Procedures](#) topic in this section for information about storing your compiled code and updating the CLASSPATH to ensure that Splice Machine can find your code.
- » Refer to the [Functions and Stored Procedure Examples](#) topic in this section for complete sample code for both a function and a stored procedure.

Note that the processes for adding functions and stored procedures to your Splice Machine database are quite similar; however, there are some important differences, so we've separated them into their own sections below.

Writing a Function in Splice Machine

Follow the steps below to write a Splice Machine database function.

1. Create a Java method

Each function maps to a Java method. For example:

```
package com.splicemachine.cs.function;

public class Functions {
    public static int addNumbers(int val1, int val2) {
        return val1 + val2;
    }
}
```

2. Create the function in the database

You can find the complete syntax for CREATE FUNCTION in the *Splice Machine SQL Reference* manual.

Here's a quick example of creating a function. In this example, `com.splicemachine.cs.function` is the package, `Functions` is the class name, and `addNumbers` is the method name:

```
CREATE FUNCTION add(val1 int, val2 int)
    RETURNS integer
    LANGUAGE JAVA
    PARAMETER STYLE JAVA
    NO SQL
    EXTERNAL NAME 'com.splicemachine.cs.function.Functions.addNumbers';
```

3. Store your compiled Jar file and update your CLASSPATH

Follow the instructions in the [Storing and Updating Functions and Stored Procedures](#) topic in this section to:

- » store your Jar file
- » update the class path so that Splice Machine can find your code when the function is called.

Invoke your function

You can invoke functions just like you would call any built-in database function. For example, if you're using the Splice Machine command line interface (*CLI*), and have created a function named add, you could use a statement like the following:

```
SELECT add(1,2) FROM SYS.SYSTABLES;
```

Writing a Stored Procedure in Splice Machine

Follow the steps below to write a Splice Machine database stored procedure.

1. Write your custom stored procedure:

Here is a very simple stored procedure that uses JDBC:

```

package org.splicetest.customprocs;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 * This class contains custom stored procedures that will be dynamically
loaded into the Splice Machine
 * database with the SQLJ jar file loading system procedures.
 *
 * @author Splice Machine
 */

public class CustomSpliceProcs {
    /**
     * Return the names for all tables in the database.
     *
     * @param rs      result set containing names of all the tables in teh dat
abase
     */
    public static void GET_TABLE_NAMES(ResultSet[] rs)
        throws SQLException
    {
        Connection conn = DriverManager.getConnection("jdbc:default:conne
ction");
        PreparedStatement pstmt = conn.prepareStatement("select * from sy
s.systables");
        rs[0] = pstmt.executeQuery();
        conn.close();
    }
}

```

You can use any Java IDE or text edit to write your code.

You can find additional examples in the [Functions and Stored Procedure Examples](#) topic in this section.

NOTE: See the information about [working with ResultSets](#) in the next section.

2. Compile your code and build a Jar file

You now need to compile your stored procedure and build a jar file for it.

You can use any Java IDE or build tool, such as *Maven* or *Ant*, to accomplish this. Alternatively, you can use the *javac* Java compiler and the *Java Archive* tool packaged with the JDK.

3. Copy the Jar file to a cluster node

Next, copy your custom Jar file to a region server (any node running an HBase region server) in your Splice Machine cluster. You can copy the file anywhere that allows the `splice>` interface to access it.

You can use any remote copying tool, such as `scp` or `ftp`. For example:

```
scp custom-splice-procs-1.0.2-SNAPSHOT.jar splice@myServer:myDir
```

See the [Storing and Updating Functions and Stored Procedures](#) topic in this section for more information.

4. Deploy the Jar file to your cluster

Deploying the Jar file requires you to install the file in your database, and to add it to your database's `CLASSPATH`. You can accomplish both of these steps by calling built-in system procedures from the `splice>` command line interpreter. For example:

```
CALL SQLJ.INSTALL_JAR(
  '/Users/splice/my-directory-for-jar-files/custom-splice-procs-2.7.0-SNA
  PSHOT.jar',
  'SPLICE.CUSTOM_SPICE_PROCS_JAR', 0);

CALL SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY(
  'derby.database.classpath', 'SPLICE.CUSTOM_SPICE_PROCS_JAR');
```

The `SQLJ.INSTALL_JAR` system procedure uploads the jar file from the local file system where `splice>` is executing into the HDFS:

- » If you are running a cluster, the Jar files are stored under the `/hbase/splicedb/jar` directory in HDFS (or MapR-FS).
- » If you are running in standalone mode, the Jar files are stored on the local file system under the `splicedb/jar` directory in the Splice install directory.

5. Register your stored procedure with Splice Machine

Register your stored procedure with the database by calling the `CREATE PROCEDURE` statement. For example:

```
CREATE PROCEDURE SPLICE.GET_TABLE_NAMES( )
  PARAMETER STYLE JAVA
  READS SQL DATA
  LANGUAGE JAVA
  DYNAMIC RESULT SETS 1
  EXTERNAL NAME 'org.splicetest.customprocs.CustomSpliceProcs.GET_TABLE_NAMES';
```

Note that after running the above `CREATE PROCEDURE` statement, your procedure will show up in the list of available procedures when you run the `Splice Machine show procedures` command.

You can find the complete syntax for `CREATE PROCEDURE` in the *Splice Machine SQL Reference* manual.

6. Run your stored procedure

You can run your stored procedure by calling it from the `splice>` prompt. For example:

```
splice> call SPLICE.GET_TABLE_NAMES();
```

7. Updating/Reloading your stored procedure

If you make changes to your procedure's code, you need to create a new Jar file and reload that into your database by calling the `SQLJ.REPLACE_JAR` system procedure:

```
CALL SQLJ.REPLACE_JAR(
    '/Users/splice/my-directory-for-jar-files/custom-splice-procs-2.7.0-SNA
    PSHOT.jar',
    'SPLICE.CUSTOM_SPICE_PROCS_JAR');
```

Working with ResultSets

Splice Machine follows the SQL-J part 1 standard for returning `ResultSets` through Java procedures. Each `ResultSet` is returned through one of the parameters passed to the java method. For example, the `resultSet` parameter in the `MY_TEST_PROC` method in our `ExampleStoredProc` class:

```
public class ExampleStoredProc {
    public static void MY_TEST_PROC(String myInput, ResultSet[] resultSet) throws SQL
Exception {
    ...
}
```

Here are a set of things you should know about `ResultSets` [] in stored procedures:

- » The `ResultSets` are returned in the order in which they were created.
- » The `ResultSets` must be open and generated from the `jdbc:default:connection` default connection. Any other `ResultSets` are ignored.
- » If you close the statement that created the `ResultSet` within the procedure's method, that closes the `ResultSet` you want. Instead, you can close the connection.
- » The Splice Machine database engine itself creates the one element `ResultSet` arrays that hold the returned `ResultSets`.
- » Although the `CREATE PROCEDURE` call allows you to specify the number of `DYNAMIC RESULT SETS`, we currently only support returning a single `ResultSet`.

Storing and Updating Splice Machine Functions and Stored Procedures

This topic describes how to store and update your compiled Java Jar (.jar) files when developing stored procedures and functions for Splice Machine.

NOTE: Jar files are not versioned: the GENERATIONID is always zero. You can view the metadata for the Jar files in the Splice data dictionary by executing this query: `select * from sys.sysfiles;`

Adding a Jar File

To add a new Jar file to your Splice Machine database, use the `splice>` command line interface to store the Jar and then update your `CLASSPATH` property so that your code can be found:

NOTE: When Splice Machine is searching for a class to load, it first searches the system `CLASSPATH`. If the class is not found in the traditional system class path, Splice Machine then searches the class path set as the value of the `derby.database.classpath` property.

1. Load your Jar file into the Splice Machine database

```
splice> CALL SQLJ.INSTALL_JAR(
    '/Users/me/dev/workspace/examples/bin/example.jar',
    'SPLICE.MY_EXAMPLE_APP', 0);
```

Please refer to the [SQLJ.INSTALL_JAR](#) topic for more information about using this system procedure. To summarize:

- » The first argument is the path on your computer to your Jar file.
- » The second argument is the name for the stored procedure Jar file in your database, in `schema.name` format.
- » The third argument is currently unused but required; use 0 as its value.

2. Update your `CLASSPATH`

You need to update your `CLASSPATH` so that Splice Machine can find your code. You can do this by using the [SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY](#) system procedure to update the `derby.database.classpath` property:

```
splice> CALL SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY(
    'derby.database.classpath',
    'SPLICE.MY_EXAMPLE_APP');
```

Note that if you've developed more than one Jar file, you can update the `derby.database.classpath` property with multiple Jars by separating the Jar file names with colons when you call the `SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY` system procedure . For example:

```
splice> CALL SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY(
    'derby.database.classpath',
    'SPLICE.MY_EXAMPLE_APP:SPLICE.YOUR_EXAMPLE');
```

Updating a Jar File

You can use the `splice>` command line interface to replace a Jar file:

1. Replace the stored Jar file

```
splice> CALL SQLJ.REPLACE_JAR(
    '/Users/me/dev/workspace/examples/bin/example.jar',
    'SPLICE.MY_EXAMPLE_APP');
```

Please refer to the [SQLJ.REPLACE_JAR](#) topic for more information about using this system procedure. To summarize:

- » The first argument is the path on your computer to your Jar file.
- » The second argument is the name for the stored procedure Jar file in your database, in `schema.name` format.

Deleting a Jar File

You can use the `splice>` command line interface to delete a Jar file:

1. Delete a stored Jar file

```
splice> CALL SQLJ.REMOVE_JAR('SPLICE.MY_EXAMPLE_APP', 0);
```

Please refer to the [SQLJ.REMOVE_JAR](#) topic for more information about using this system procedure. To summarize:

- » The first argument is the name for the stored procedure Jar file in your database, in schema.name format.
- » The second argument is currently unused but required; use 0 as its value.

NOTE: The Jar file operations (the [SQLJ.REMOVE_JAR](#) system procedures) are not executed within transactions, which means that committing or rolling back a transaction will not have any impact on these operations.

Examples of Splice Machine Functions and Stored Procedures

This topic walks you through creating, storing, and using a sample database function and a sample database stored procedure, in these sections:

- » [Creating and Using a Sample Function in Splice Machine](#)
- » [Creating and Using a Sample Stored Procedure in Splice Machine](#)

Creating and Using a Sample Function in Splice Machine

This section walks you through creating a sample function named `word_limiter` that limits the number of words in a string; for example, given this sentence:

Today is a wonderful day and I am looking forward to going to the beach.

If you tell `word_limiter` to return the first five words in the sentence, the returned string would be:

Today is a wonderful day

Follow these steps to define and use the `word_limiter` function:

1. Define inputs and outputs

We have two inputs:

- » the sentence that we want to limit
- » the number of words to which we want to limit the output

The output is a string that contains the limited words.

2. Create the shell of our Java class.

We create a class named `ExampleStringFunctions` in the package `com.splicemachine.examples`.

```
package com.splicemachine.example;

public class ExampleStringFunctions {...}
```

3. Create the `wordLimiter` static method

This method contains the logic for returning the first n number of words:

```

package com.splicemachine.example;

public class ExampleStringFunctions {
    /**
     * Truncates a string to the number of words specified. An input of
     * "Today is a wonderful day and I am looking forward to going to th
     e beach.", 5
     * will return "Today is a wonderful day".
     *
     * @param inboundSentence
     * @param numberOfWords
     * @return
     */
    public static String wordLimiter(String inboundSentence, int numberOf
Words) {
        String truncatedString = "";
        if(inboundSentence != null) {
            String[] splitBySpace = inboundSentence.split("\\s+");
            if(splitBySpace.length = numberOfWords) {
                truncatedString = inboundSentence;
            } else {
                StringBuilder sb = new StringBuilder();
                for(int i=0; i<numberOfWords; i++) {
                    if(i > 0) sb.append(" ");
                    sb.append(splitBySpace[i]);
                }
                truncatedString = sb.toString();
            }
        }
        return truncatedString;
    }
}

```

4. Compile the class and store the jar file

After you compile your class, make sure that the jar file is in a directory in your classpath, so that it can be found. You can find more information about this in the [Storing and Updating Functions and Stored Procedures](#) topic in this section.

If you're using the standalone version of Splice Machine, you can use the following command line interface command:

```

splice> CALL SQLJ.INSTALL_JAR(
    '/Users/me/dev/workspace/examples/bin/example.jar',
    'SPLICE.MY_EXAMPLE_APP', 0);

```

You must also update the database class path:

```
splice> CALL SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY(
    'derby.database.classpath',
    'SPLICE.MY_EXAMPLE_APP');
```

5. Define the function in Splice Machine

You can find the complete syntax for [CREATE FUNCTION](#) in the *Splice Machine SQL Reference* manual. For our example function, we enter the following command at the `splice>` prompt:

```
splice> CREATE FUNCTION WORD_LIMITER(
    MY_SENTENCE VARCHAR(9999),
    NUM_WORDS INT) RETURNS VARCHAR(9999)
LANGUAGE JAVA
PARAMETER STYLE JAVA
NO SQL
EXTERNAL NAME 'com.splicemachine.example.ExampleStringFunctions.wordLimiter';
```

6. Run the function

You can run the function with this syntax:

```
splice> SELECT WORD_LIMITER(
    'Today is a wonderful day and I am looking forward to going to the beach.', 5)
FROM SYSIBM.SYSDUMMY1;
```

Creating and Using a Sample Stored Procedure in Splice Machine

In this section , we create a stored procedure named `GET_INVENTORY_FOR_SKU` that retrieves all of the inventory records for a specific product by using the product's sku code. The input to this procedure is the sku code, and the output is a resultset of records from the inventory table.

Follow these steps to define and use the `GET_INVENTORY_FOR_SKU` function:

1. Create and populate the inventory table

Connect to Splice Machine and create the following table from the command prompt or from an SQL client, using the following statements:

```

CREATE TABLE INVENTORY (
    SKU_CODE VARCHAR(30),
    WAREHOUSE BIGINT,
    QUANTITY BIGINT
);

INSERT INTO INVENTORY VALUES ('ABC123',1,50),('ABC123',2,100),('ABC12
3',3,60),('XYZ987',1,20),('XYZ321',2,0);

```

2. Create the shell of our Java class.

We create a class named `ExampleStringFunctions` in the package `com.splicemachine.examples`.

```

package com.splicemachine.example;

public class ExampleStoredProcedure{...
}

```

3. Create the `getSkuInventory` static method

This method contains the logic for retrieving inventory records for the specified sku.

```

package com.splicemachine.example;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
public class ExampleStoredProcedure {
    public static void getSkuInventory(String skuCode, ResultSet[] results
et) throws SQLException {
        try {
            Connection con = DriverManager.getConnection("jdbc:default:con
nection");
            String sql = "select * from INVENTORY " + "where SKU_CODE =
?";
            PreparedStatement ps = con.prepareStatement(sql);
            ps.setString(1, skuCode);
            resultSet[0] = ps.executeQuery();
        } catch (SQLException e) {
            throw e;
        }
    }
}

```

4. Compile the class and store the jar file

After you compile your class, make sure that the jar file is in a directory in your `classpath`, so that it can be found. You can find more information about this in the [Storing and Updating Functions and Stored Procedures](#) topic in this section.

If you're using the standalone version of Splice Machine, you can use the following command line interface command:

```
splice> CALL SQLJ.INSTALL_JAR('/Users/me/dev/workspace/examples/bin/example.jar', 'SPLICE.GET_SKU_INVENTORY', 0);
```

You must also update the database class path:

```
splice> CALL SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY('derby.database.classpath', 'SPLICE.GET_SKU_INVENTORY');
```

5. Define the procedure in Splice Machine

For our procedure, we enter the following command at the `splice>` prompt:

```
splice> CREATE PROCEDURE GET_INVENTORY_FOR_SKU(SKU_CODE VARCHAR(30))
LANGUAGE JAVA
PARAMETER STYLE JAVA
READS SQL DATA
EXTERNAL NAME 'com.splicemachine.example.ExampleStoredProcedure.getSkuInventory';
```

6. Run the stored procedure

You can run the procedure with this syntax:

```
splice> call GET_INVENTORY_FOR_SKU('ABC123');
```

Tuning and Debugging Query Performance

This section contains the following topics to help you optimize your Splice Machine database queries:

Topic	Describes
Optimizing Queries	Gets you started with optimizing your queries for Splice Machine.
Using Statistics	Using our statistics gathering facility to help optimize queries.
Explain Plan	The Explain Plan feature, which allows you to examine the execution plan for a query without executing the query.
Logging	The Splice Machine logging facility.
Debugging	Describes the parameter values to use when using debugger software with Splice Machine.



For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](#).

Optimizing Splice Machine Queries

This topic introduces you to Splice Machine query optimization techniques, including information about executing SQL expressions without a table context, and using optimization hints.

Introduction to Query Optimization

Here are a few mechanisms you can use to optimize your Splice Machine queries:

Optimization Mechanism	Description
Use Explain Plan	You can use the Splice Machine Explain Plan facility to display the execution plan for a statement without actually executing the statement. You can use Explain Plan to help determine options such as which join strategy to use or which index to select. See the About Explain Plan topic.
Use Statistics	Your database administrator can refine which statistics are collected on your database, which in turn enhances the operation of the query optimizer. See the Using Statistics topic.
Use WHERE clauses	Use a WHERE clause in your queries to restrict how much data is scanned.
Use indexes	You can speed up a query by having an index on the criteria used in the WHERE clause, or if the WHERE clause is using a primary key. Composite indexes work well for optimizing queries.
Use Splice Machine <i>query hints</i>	The Splice Machine query optimizer allows you to provide hints to help in the optimization process.

Using Select Without a Table

Sometimes you want to execute SQL scalar expressions without having a table context. For example, you might want to create a query that evaluates an expression and returns a table with a single row and one column. Or you might want to evaluate a list of comma-separated expressions and return a table with a single row and multiple columns one for each expression.

In Splice Machine, you can execute queries without a table by using the `sysibm.sysdummy1` dummy table. Here's the syntax:

```
select expression FROM sysibm.sysdummy1
```

And here's an example:

```
splice> select 1+ 1 from sysibm.sysdummy1;
```

Using Splice Machine Query Hints

You can use *hints* to help the Splice Machine query interface optimize your database queries.

NOTE: The Splice Machine optimizer is constantly being improved, and new hint types sometimes get added. One recent addition is the ability to specify that a query should be run on (or not on) Spark, if possible.

Types of Hints

There are different kinds of hints you can supply, each of which is described in a section below; here's a summary:

Hint Type	Examples	Used to Specify
Index	--splice-properties index=my_index	Which index to use or not use
Join Order	--splice-properties joinOrder=fixed	Which join order to use for two tables
Join Strategy	--splice-properties joinStrategy=sortmerge	How a join is processed (in conjunction with the Join Order hint)
Pinned Table	--splice-properties pin=true	That you want the pinned (cached in memory) version of a table used in a query
Spark	--splice-properties useSpark=true	That you want a query to run (or not run) on Spark
Delete	--splice-properties bulkDeleteDirectory=' /path '	That you are deleting a large amount of data and want to bypass the normal write pipeline to speed up the deletion.

Including Hints in Your Queries

Hints MUST ALWAYS be at the end of a line, meaning that you must always terminate hints with a newline character.

You cannot add the semicolon that terminates the command immediately after a hint; the semicolon must go on the next line, as shown in the examples in this topic.



Many of the examples in this section show usage of hints on the `splice>` command line. Follow the same rules when using hints programmatically.

Hints can be used in two locations: after a table identifier or after a `FROM` clause. Some hint types can be used after a table identifier, and some can be used after a `FROM` clause:

Hint after a:	Hint types	Example
Table identifier	<code>index</code> <code>joinStrategy</code> <code>pin</code> <code>useSpark</code> <code>bulkDeleteDirectory</code>	<pre>SELECT * FROM member_info m, rewards r, points p --SPLICE-PROPERTIES index=ie_point WHERE...</pre>
A <code>FROM</code> clause	<code>joinOrder</code>	<pre>SELECT * FROM --SPLICE-PROPERTIES joinOrder=fixed mytable1 e, mytable2 t WHERE e.id = t.parent_id;</pre>

This example shows proper placement of the hint and semicolon when the hint is at the end of statement:

```
SELECT * FROM my_table --splice-properties index=my_index;
```

If your command is broken into multiple lines, you still must add the hints at the end of the line, and you can add hints at the ends of multiple lines; for example:

```
SELECT * FROM my_table_1 --splice-properties index=my_index
, my_table_2 --splice-properties index=my_index_2
WHERE my_table_1.id = my_table_2.parent_id;
```

NOTE: In the above query, the first command line ends with the first index hint, because hints must always be the last thing on a command line. That's why the comma separating the table specifications appears at the beginning of the next line.

Index Hints

Use *index hints* to tell the query interface how to use certain indexes for an operation.

To force the use of a particular index, you can specify the index name; for example:

```
splice> SELECT * FROM my_table --splice-properties index=my_index
> ;
```

To tell the query interface to not use an index for an operation, specify the null index. For example:

```
splice> SELECT * FROM my_table --splice-properties index=null
> ;
```

And to tell the query interface to use specific indexes on specific tables for an operation, you can add multiple hints. For example:

```
splice> SELECT * FROM my_table_1 --splice-properties index=my_index
> , my_table_2 --splice-properties index=my_index_2
> WHERE my_table_1.id = my_table_2.parent_id;
```

Important Note About Placement of Index Hints

Each index hint in a query **MUST** be specified alongside the table containing the index, or an error will occur.

For example, if we have a table named `points` with an index named `ie_point` and another table named `rewards` with an index named `ie_rewards`, then this hint works as expected:

```
SELECT * FROM member_info m,
      rewards r,
      points p   --SPLICE-PROPERTIES index=ie_point
WHERE...
```

But the following hint will generate an error because `ie_rewards` is not an index on the `points` table.

```
SELECT * FROM
      member_info m,
      rewards r,
      points p   --SPLICE-PROPERTIES index=ie_rewards
WHERE...
```

JoinOrder Hints

Use `JoinOrder` hints to tell the query interface in which order to join two tables. You can specify these values for a `JoinOrder` hint:

- » Use `joinOrder=FIXED` to tell the query optimizer to order the table join according to how where they are named in the `FROM` clause.
- » Use `joinOrder=UNFIXED` to specify that the query optimizer can rearrange the table order.

NOTE: `joinOrder=UNFIXED` is the default, which means that you don't need to specify this hint to allow the optimizer to rearrange the table order.

Here are examples:

Hint	Example
joinOrder=FIXED	splice> SELECT * FROM --SPLICE-PROPERTIES joinOrder=fixed> mytable1 e, mytable2 t> WHERE e.id = t.parent_id;
joinOrder=UNFIXED	splice> SELECT * from --SPLICE-PROPERTIES joinOrder=unfixed> mytable1 e, mytable2 t WHERE e.id = t.parent_id;

JoinStrategy Hints

You can use a `JoinStrategy` hint in conjunction with a `joinOrder` hint to tell the query interface how to process a join. For example, this query specifies that the `SORTMERGE` join strategy should be used:

```
SELECT * FROM      --SPLICE-PROPERTIES joinOrder=fixed
    mytable1 e, mytable2 t --SPLICE-PROPERTIES joinStrategy=SORTMERGE
    WHERE e.id = t.parent_id;
```

And this uses a `joinOrder` hint along with two `joinStrategy` hints:

```
SELECT *
  FROM --SPLICE-PROPERTIES joinOrder=fixed
    keyword k
  JOIN campaign c  --SPLICE-PROPERTIES joinStrategy=NESTEDLOOP
    ON k.campaignid = c.campaignid
  JOIN adgroup g  --SPLICE-PROPERTIES joinStrategy=NESTEDLOOP
    ON k.adgroupid = g.adgroupid
  WHERE adid LIKE '%us_gse%'
```

You can specify these join strategies:

JoinStrategy Value	Strategy Description
BROADCAST	Read the results of the Right Result Set (<i>RHS</i>) into memory, then for each row in the left result set (<i>LHS</i>), perform a local lookup to determine the right side of the join. BROADCAST will only work on equijoin (=) predicates that do not include a function call.

MERGE	Read the Right and Left result sets simultaneously in order and join them together as they are read. MERGE joins require that both the left and right result sets be sorted according to the join keys. MERGE requires an equijoin predicate that does not include a function call.
NESTEDLOOP	For each row on the left, fetch the values on the right that match the join. NESTEDLOOP is the only join that can work with any join predicate of any type; however this type of join is generally very slow.
SORTMERGE	Re-sort both the left and right sides according to the join keys, then perform a MERGE join on the results. SORTMERGE requires an equijoin predicate with no function calls.

Pinned Table Hint

You can use the `pin` hint to specify to specify that you want a query to run against a pinned version of a table.

```
splice> PIN TABLE myTable;splice> SELECT COUNT(*) FROM my_table --splice-properties
pin=true> ;
```

You can read more about pinning tables in the [PIN TABLE](#) statement topic.

Spark Hints

You can use the `useSpark` hint to specify to the optimizer that you want a query to run on (or not on) Spark. The Splice Machine query optimizer automatically determines whether to run a query through our Spark engine or our HBase engine, based on the type of query; you can override this by using a hint:

NOTE: The Splice Machine optimizer uses its estimated cost for a query to decide whether to use spark. If your statistics are out of date, the optimizer may end up choosing the wrong engine for the query.

```
splice> SELECT COUNT(*) FROM my_table --splice-properties useSpark=true> ;
```

You can also specify that you want the query to run on HBase and not on Spark. For example:

```
splice> SELECT COUNT(*) FROM your_table --splice-properties useSpark=false> ;
```

Delete Hints

You can use the `bulkDeleteDirectory` hint to specify that you want to use our bulk delete feature to optimize the deletion of a large amount of data. Similar to our bulk import feature, bulk delete generates HFiles, which allows us to bypass the Splice Machine write pipeline and HBase write path when performing the deletion. This can significantly speed up the deletion process.

You need to specify the directory to which you want the temporary HFiles written; you must have write permissions on this directory to use this feature. If you're specifying an S3 bucket on AWS, please review our [Configuring an S3 Bucket for Splice Machine Access](#) tutorial before proceeding.

```
splice> DELETE FROM my_table --splice-properties bulkDeleteDirectory='/bulkFilesPath'  
;  
;
```

NOTE: We recommend performing a major compaction on your database after deleting a large amount of data; you should also be aware of our new [SYSCS_UTIL.SET_PURGE_DELETED_ROWS](#) system procedure, which you can call before a compaction to specify that you want the data physically (not just logically) deleted during compaction.

Using Statistics with Splice Machine

This topic introduces how to use database statistics with Splice Machine. Database statistics are a form of metadata (data about data) that assists the Splice Machine query optimizer; the statistics help the optimizer select the most efficient approach to running a query, based on information that has been gathered about the tables involved in the query.

This topic describes how to:

- » [Collecting statistics](#) for a schema or table in your database
- » [Dropping statistics](#) for a schema
- » [Enable and disable collection of statistics](#) on specific columns in tables in your database
- » [View collected statistics](#)

Statistics are inexact; in fact, some statistics like table cardinality are estimated using advanced algorithms, due to the resources required to compute these values. It's important to keep this in mind when basing design decisions on values in database statistics tables.

It's also important to note that the statistics for your database are not automatically refreshed when the data in your database changes, which means that when you query a statistical table or view, the results you see may not exactly match the data in the actual tables.

Collecting Statistics

You can collect statistics on a schema or table using the `splice> Analyze` command.

You can also use the `SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS` procedure to collect statistics on an entire schema, including every table in the schema. For example:

```
splice> CALL SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS( 'SPLICEBBALL', false );
```

During statistical collection:

- » Statistics are automatically collected on columns in a primary key, and on columns that are indexed. These are called *keyed columns*.
- » Statistics are also collected on columns for which you have enabled statistics collection, as described in the next section, [Enabling and Disabling Statistics](#).

Once collection of statistics has completed, the Splice Machine query optimizer will automatically begin using the updated statistics to optimize query execution plans.

When Should You Collect Statistics?

We advise that you collect statistics after you have:

- » Created an index on a table.
- » Modified a significant number of rows in a table with update, insert, or delete operations.

NOTE: A general rule-of-thumb is that you should collect statistics after modifying more than 10% of data.

On Which Columns Should You Collect Statistics?

By default, Splice Machine collects statistics on all columns in a table.

To reduce the operational cost of analyzing large tables (such as fact tables), you can tell Splice Machine to not collect statistics on certain columns by running the `SYSCS_UTIL.DISABLE_COLUMN_STATISTICS` built-in system procedure:

```
SYSCS_UTIL.DISABLE_COLUMN_STATISTICS( schema, table, column);
```

Splice Machine strongly recommends that you *always* collect statistics on small tables, such as a table that has hundreds of rows on each region server.

How to Determine if You Should Collect or Drop Statistics

You can use [Explain Plan](#) in your development or test environment to determine how dropping or collecting statistics changes the execution plan for a query.

Dropping Statistics

If you subsequently wish to drop statistics for a schema, you can use the `SYSCS_UTIL.DROP_SCHEMA_STATISTICS` procedure to drop statistics for an entire schema. For example:

```
splice> CALL SYSCS_UTIL.DROP_SCHEMA_STATISTICS('SPLICEBBALL');
```

Enabling and Disabling Statistics on Specific Columns

When you collect statistics, Splice Machine automatically collects statistics on keyed columns, which are columns in a primary key and columns that are indexed.

Please review the recommendations and restrictions regarding which columns should or should not have statistics collection enabled, as noted below in the [Selecting Columns for Statistics Collection](#) subsection.

You can also explicitly enable statistics collection on specific columns in tables using the `SYSCS_UTIL.ENABLE_COLUMN_STATISTICS` procedure. For example:

```
CALL SYSCS_UTIL.ENABLE_COLUMN_STATISTICS('SPLICEBBALL', 'Players', 'Birthdate');
```

Disabling Statistics Collection on a Column

If you subsequently wish to disable collection of statistics on a specific column in a table, use the `SYSCS_UTIL.DISABLE_COLUMN_STATISTICS` procedure:

```
splice> CALL SYSCS_UTIL.DISABLE_COLUMN_STATISTICS('SPLICEBBALL', 'Players', 'Birthdate');
```

Once you've enabled or disabled statistics collection for one or more table columns, you should update the query optimizer by [collecting statistics](#) on the table or schema.

Selecting Columns for Statistics Collection

You can only collect statistics on columns containing data that can be ordered. This includes all numeric types, Boolean values, some CHAR and BIT data types, and date and timestamp values.

When selecting columns on which statistics should be collected, keep these ideas in mind:

- » The process of collecting statistics requires both memory and compute time to complete; the more statistics you collect, the longer it takes and the more of your computing resources that it uses.
- » You *should collect statistics* for any column that is used as a predicate in a query.
- » You *should collect statistics* for any column that is used in a `select distinct`, `Group by`, `order by`, or `join` clause.
- » You *do not need to enable statistics* for columns that are merely carried through the computation; however, doing so may improve heap size estimations, which in turn can make broadcast joins more likely to be chosen.

As you can see, selecting columns for statistics is a tradeoff between the resources required to collect the statistics, and the improvements in optimization that result from having the statistics collected.

Viewing Collected Statistics

Splice Machine provides two system tables you can query to view the statistics that have been collected for your database:

- » Query the [SYSTABLESTATISTICS System Table](#) to view statistics for specific tables
- » Query the [SYSCOLUMNSTATISTICS System Table](#) to view statistics for specific table columns

See Also

- » [Splice Machine Data Assignments and Comparisons](#)
- » [SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS](#)
- » [SYSCS_UTIL.DISABLE_COLUMN_STATISTICS](#)
- » [SYSCS_UTIL.DROP_SCHEMA_STATISTICS](#)
- » [SYSCS_UTIL.ENABLE_COLUMN_STATISTICS](#)
- » [SYSCOLUMNSTATISTICS System Table](#)
- » [SYSTABLESTATISTICS System Table](#)

About Explain Plan

You can use the `explain` command to display what the execution plan will be for a statement without executing the statement. This topic presents and describes several examples.

Using Explain Plan to See the Execution Plan for a Statement

To display the execution plan for a statement without actually executing the statement, use the `explain` command on the statement:

```
splice> explain Statement;
```

Statement

An SQL statement.

Explain Plan and DDL Statements

SQL Data Definition Language (DDL) statements have no known cost, and thus do not require optimization. Because of this, the `explain` command does not work with DDL statements; attempting to `explain` a DDL statement such as `CREATE TABLE` will generate a syntax error. You **cannot** use `explain` with any of the following SQL statements:

- » `ALTER`
- » `CREATE ...` (any statement that starts with `CREATE`)
- » `DROP ...` (any statement that starts with `DROP`)
- » `GRANT`
- » `RENAME ...` (any statement that starts with `RENAME`)
- » `REVOKE`
- » `TRUNCATE TABLE`

Explain Plan Output

When you run the `explain` command, it displays output in a tree-structured format:

- » The first row in the output summarizes the plan
- » Each row in the output represents a node in the tree.
- » For join nodes, the right side of the join is displayed first, followed by the left side.
- » Child node rows are indented and prefixed with '`->`'.

The first node in the plan output contains these fields:

Field	Description
<code>n=number</code>	The number of nodes.
<code>rows=number</code>	The number of output rows.
<code>updateMode=[mode]</code>	The update mode of the statement.
<code>engine=[engineType]</code>	<code>engineType=Spark</code> means this query will be executed by our OLAP engine. <code>engineType=control</code> means this query will be executed by our OLTP engine.

Each node row in the output contains the following fields:

Field	Description
<code>[node_label]</code>	The name of the node
<code>n=number</code>	The result set number. This is primarily used internally, and can also be used to determine the relative ordering of optimization.
<code>totalCost=number</code>	The total cost to perform this operation. This is computed <i>as if the operation is at the top of the operation tree</i> .
<code>processingCost=number</code>	The cost to process all data in this node. Processing includes everything except for reading the final results over the network.
<code>transferCost=number</code>	The cost to send the final results over the network to the control node.
<code>outputRows=number</code>	The total number of rows that are output.

Field	Description
outputHeapSize= <i>number unit</i>	The total size of the output result set, and the unit in which that size is expressed, which is one of: <ul style="list-style-type: none"> » B » KB » MB » GB » TB
partitions== <i>number</i>	The number of partitions involved. <i>Partition</i> is currently equivalent to <i>Region</i> ; however, this will not necessarily remain true in future releases.

For example:

```
splice> explain select * from sys.systables t, sys.sysschemas s
      where t.schemaid = s.schemaid;

Plan
-----
-----Cursor(n=5,rows=20,updateMode=READ_ONLY (1),engine=control)
 -> ScrollInsensitive(n=4,totalCost=21.728,outputRows=20,outputHeapSize=6.641 KB,partitions=1)
     -> BroadcastJoin(n=3,totalCost=12.648,outputRows=20,outputHeapSize=6.641 KB,partitions=1,preds=[(T.SCHEMAID[4:4] = S.SCHEMAID[4:8])])
         -> TableScan[SYSSCHEMAS(32)](n=2,totalCost=4.054,outputRows=20,outputHeapSize=6.641 KB,partitions=1)
             -> TableScan[SYSTABLES(48)](n=1,totalCost=4.054,outputRows=20,outputHeapSize=3.32 KB,partitions=1)

5 rows selected
```

The next topic, [Explain Plan Examples](#), contains a number of examples, annotated with notes to help you understand the output of each.

See Also

- » [Explain Plan Examples](#)

Explain Plan Examples

This topic contains the following examples of using the `explain` command to display the execution plan for a statement, including the following examples:

- » [TableScan Examples](#)
- » [IndexScan Examples](#)
- » [Projection and Restriction Examples](#)
- » [Index Lookup](#)
- » [Join Example](#)
- » [Union Example](#)
- » [Order By Example](#)
- » [Aggregation Operation Examples](#)
- » [Subquery Example](#)



The format and meaning of the common fields in the output plans shown here is found in the previous topic, [About Explain Plan](#).

TableScan Examples

This example show a plan for a TableScan operation that has no qualifiers, known as a *raw scan*:

```
splice> explain select * from sys.systables;Plan
-----
-----  

Cursor(n=3,rows=20,updateMode=READ_ONLY (1),engine=control)
  -> ScrollInsensitive(n=2,totalCost=8.594,outputRows=20,outputHeapSize=3.32 KB,partitions=1)
    -> TableScan[SYSTABLES(48)](n=1,totalCost=4.054,outputRows=20,outputHeapSize=3.32 KB,partitions=1)

3 rows selected
```

This example show a plan for a TableScan operation that does have qualifiers::

```
splice> explain select * from sys.systables --SPLICE-PROPERTIES index=NULL where tableName='SYSTABLES';Plan
-----
-----Cursor(n=3,rows=18,updateMode=READ_ONLY (1),engine=control)
-> ScrollInsensitive(n=2,totalCost=8.54,outputRows=18,outputHeapSize=2.988 KB,partitions=1)
--> TableScan[SYSTABLES(48)](n=1,totalCost=4.054,outputRows=18,outputHeapSize=2.988 KB,partitions=1,preds=[(TABLENAME[0:2] = SYSTABLES)])
3 rows selected
```

Nodes

- » The plan labels this operation as TableScan [*tableId(conglomerateId)*]:
 - » *tableId* is the name of the table, in the form schemaName '.' tableName.
 - » *conglomerateId* is an ID that is unique to every HBase table; this value is used internally, and can be used for certain administrative tasks
- » The preds field includes qualifiers that were pushed down to the base table.

IndexScan Examples

This example show a plan for an IndexScan operation that has no predicates:

```
splice> explain select tablename from sys.systables; --covering index
Plan
-----
-----Cursor(n=3,rows=20,updateMode=READ_ONLY (1),engine=control)
-> ScrollInsensitive(n=2,totalCost=8.31,outputRows=20,outputHeapSize=560 B,partitions=1)
--> IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=4.054,outputRows=20,outputHeapSize=560 B,partitions=1,baseTable=SYSTABLES(32))
3 rows selected
```

This example shows a plan for an IndexScan operation that contains predicates:

```
splice> explain select tablename from sys.systables --SPLICE-PROPERTIES index=SYSTABLES_INDEX1
      where tablename = 'SYSTABLES';

Plan
-----
-----Cursor(n=3,rows=18,updateMode=READ_ONLY (1),engine=control)
 -> ScrollInsensitive(n=2,totalCost=8.272,outputRows=18,outputHeapSize=432 B,partitions=1)
     -> IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=4.049,outputRows=18,outputHeapSize=432 B,partitions=1,baseTable=SYSTABLES(48),preds=[(TABLENAME[0:1] = SYSTABLES)])
```

3 rows selected

Nodes

- » The plan labels this operation as `IndexScan[indexId(conglomerateId)]`:
 - » `indexId` is the name of the index
 - » `conglomerateId` is an ID that is unique to every HBase table; this value is used internally, and can be used for certain administrative tasks
- » The `preds` field includes qualifiers that were pushed down to the base table.

Projection and Restriction Examples

This example show a plan for a Projection operation:

```
splice> explain select tablename || 'hello' from sys.systables;

Plan
-----
-----Cursor(n=4,rows=20,updateMode=READ_ONLY (1),engine=control)
 -> ScrollInsensitive(n=3,totalCost=8.302,outputRows=20,outputHeapSize=480 B,partitions=1)
     -> ProjectRestrict(n=2,totalCost=4.054,outputRows=20,outputHeapSize=480 B,partitions=1)
         -> IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=4.054,outputRows=20,outputHeapSize=480 B,partitions=1,baseTable=SYSTABLES(48))

4 rows selected
```

This example shows a plan for a Restriction operation:

```

splice> explain select tablename from sys.systables
      where tablename like '%SYS%';

Plan
-----
-----Cursor(n=4,rows=10,updateMode=READ_ONLY (1),engine=control)
 -> ScrollInsensitive(n=3,totalCost=8.178,outputRows=10,outputHeapSize=240 B,partitions=1)
     -> ProjectRestrict(n=2,totalCost=4.054,outputRows=10,outputHeapSize=240 B,partitions=1,preds=[like(TABLENAME[0:1], %SYS%)])
         -> IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=4.054,outputRows=20,outputHeapSize=240 B,partitions=1,baseTable=SYSTABLES(48))

4 rows selected

```

Nodes

- » The plan labels both projection and restriction operations as `ProjectRestrict`. which can contain both *projections* and *non-qualifier restrictions*. A *non-qualifier restriction* is a predicate that cannot be pushed to the underlying table scan.

Index Lookup

This example shows a plan for an `IndexLookup` operation:

```

splice> explain select * from SYS.SYSTABLES --SPLICE-PROPERTIES index=SYSTABLES_INDEX1
where tablename = 'SYSTABLES';;

Plan
-----
-----Cursor(n=4,rows=18,updateMode=READ_ONLY (1),engine=control)
 -> ScrollInsensitive(n=3,totalCost=177.265,outputRows=18,outputHeapSize=921.586 KB,partitions=1)
     -> IndexLookup(n=2,totalCost=78.715,outputRows=18,outputHeapSize=921.586 KB,partitions=1)
         -> IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=6.715,outputRows=18,outputHeapSize=921.586 KB,partitions=1,baseTable=SYSTABLES(48),preds=[(TABLENAME[1:2] = SYSTABLES)])

```

Nodes

- » The plan labels the operation as `IndexLookup`; you may see this labeled as an `IndexToBaseRow` operation elsewhere.
- » Plans for `IndexLookup` operations do not contain any special fields.

Join Example

This example shows a plan for a Join operation:

```
splice> explain select * from sys.systables t, sys.sysschemas s
      where t.schemaid = s.schemaid;

Plan
-----
-----Cursor(n=5,rows=20,updateMode=READ_ONLY (1),engine=control)
-> ScrollInsensitive(n=4,totalCost=21.728,outputRows=20,outputHeapSize=6.641 KB,partitions=1)
    -> BroadcastJoin(n=3,totalCost=12.648,outputRows=20,outputHeapSize=6.641 KB,partitions=1,preds=[(T.SCHEMAID[4:4] = S.SCHEMAID[4:8])])
        -> TableScan[SYSSCHEMAS(32)](n=2,totalCost=4.054,outputRows=20,outputHeapSize=6.641 KB,partitions=1)
            -> TableScan[SYSTABLES(48)](n=1,totalCost=4.054,outputRows=20,outputHeapSize=3.32 KB,partitions=1)

5 rows selected
```

Nodes

- » The plan labels the operation using the *join type* followed by `Join`; the possible values are:
 - » `BroadcastJoin`
 - » `MergeJoin`
 - » `MergeSortJoin`
 - » `NestedLoopJoin`
 - » `OuterJoin`
- » The plan may include a `preds` field, which lists the join predicates.
- » `NestedLoopJoin` operations do not include a `preds` field; instead, the predicates are listed in either a `ProjectRestrict` or in the underlying scan.
- » The right side of the *Join* operation is listed first, followed by the left side of the join.

Outer Joins

An *outer join* does not display it as a separate strategy in the plan; instead, it is treated a *postfix* for the strategy that's used. For example, if you are using a Broadcast join, and it's a left outer join, then you'll see `BroadcastLeftOuterJoin`. Here's an example:

```

explain select s.schemaname,t.tablename from sys.sysschemas s left outer join sys.sys
tables t
> on s.schemaid = t.schemaid;
Plan
-----
Cursor(n=6,rows=20,updateMode=READ_ONLY (1),engine=control)
-> ScrollInsensitive(n=5,totalCost=348.691,outputRows=20,outputHeapSize=2 MB,part
itions=1)
-> ProjectRestrict(n=4,totalCost=130.579,outputRows=20,outputHeapSize=2 MB,part
itions=1)
-> BroadcastLeftOuterJoin(n=3,totalCost=130.579,outputRows=20,outputHeapSiz
e=2 MB,partitions=1,preds=[(S.SCHEMAID[4:1] = T.SCHEMAID[4:4])])
-> IndexScan[SYSTABLES_INDEX1(145)](n=2,totalCost=7.017,outputRows=20,output
HeapSize=2 MB,partitions=1,baseTable=SYSTABLES(48))
-> TableScan[SYSSCHEMAS(32)](n=1,totalCost=7.516,outputRows=20,outputHeapSi
ze=1023.984 KB,partitions=1)

```

Union Example

This example shows a plan for a Union operation:

```

splice> explain select tablename from sys.systables t union all select schemaname fr
om sys.sysschemas;

Plan
-----
Cursor(n=5,rows=40,updateMode=READ_ONLY (1),engine=control)
-> ScrollInsensitive(n=4,totalCost=16.668,outputRows=40,outputHeapSize=1.094 KB,p
artitions=1)
-> Union(n=3,totalCost=12.356,outputRows=40,outputHeapSize=1.094 KB,partition
s=1)
-> IndexScan[SYSSCHEMAS_INDEX1(209)](n=2,totalCost=4.054,outputRows=20,output
HeapSize=1.094 KB,partitions=1,baseTable=SYSSCHEMAS(32))
-> IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=4.054,outputRows=20,outputH
eapSize=480 B,partitions=1,baseTable=SYSTABLES(48))

5 rows selected

```

Nodes

- » The right side of the Union is listed first, followed by the left side of the union,

Order By Example

This example shows a plan for an order by operation:

```
splice> explain select tablename from sys.systables order by tablename desc;

Plan
-----
-----Cursor(n=4,rows=20,updateMode=READ_ONLY (1),engine=control)
 -> ScrollInsensitive(n=3,totalCost=16.604,outputRows=20,outputHeapSize=480 B,partitions=1)
     -> OrderBy(n=2,totalCost=12.356,outputRows=20,outputHeapSize=480 B,partitions=1)
         -> IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=4.054,outputRows=20,outputHeapSize=480 B,partitions=1,baseTable=SYSTABLES(48))

4 rows selected
```

Nodes

- » The plan labels this operation as OrderBy.

Aggregation Operation Examples

This example show a plan for a grouped aggregate operation:

```
splice> explain select tablename, count(*) from sys.systables group by tablename;

Plan
-----
-----Cursor(n=6,rows=20,updateMode=READ_ONLY (1),engine=control)
 -> ScrollInsensitive(n=5,totalCost=12.568,outputRows=20,outputHeapSize=480 B,partitions=16)
     -> ProjectRestrict(n=4,totalCost=8.32,outputRows=20,outputHeapSize=480 B,partitions=16)
         -> GroupBy(n=3,totalCost=4.054,outputRows=20,outputHeapSize=480 B,partitions=1)
             -> ProjectRestrict(n=2,totalCost=4.054,outputRows=20,outputHeapSize=480 B,partitions=1)
                 -> IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=4.054,outputRows=20,outputHeapSize=480 B,partitions=1,baseTable=SYSTABLES(48))

6 rows selected)
```

This example shows a plan for a scalar aggregate operation:

```
splice> explain select count(*) from sys.systables;

Plan
-----
-----Cursor(n=6,rows=1,updateMode=READ_ONLY (1),engine=control)
 -> ScrollInsensitive(n=5,totalCost=8.797,outputRows=1,outputHeapSize=0 B,partitions=1)
     -> ProjectRestrict(n=4,totalCost=4.257,outputRows=1,outputHeapSize=0 B,partitions=1)
         -> GroupBy(n=3,totalCost=4.054,outputRows=20,outputHeapSize=3.32 KB,partition
s=1)
             -> ProjectRestrict(n=2,totalCost=4.054,outputRows=20,outputHeapSize=3.32 K
B,partitions=1)
                 -> IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=4.054,outputRows=20,out
putHeapSize=3.32 KB,partitions=1,baseTable=SYSTABLES(48))

6 rows selected
```

Nodes

- » The plan labels both grouped and scaled aggregate operations as GroupBy.

Subquery Example

This example shows a plan for a SubQuery operation:

```
splice> explain select tablename, (select tablename from sys.systables t2 where t2.tablename = t.tablename)from sys.systables t;

Plan
-----
-----Cursor(n=6,rows=20,updateMode=READ_ONLY (1),engine=control)
 -> ScrollInsensitive(n=5,totalCost=8.302,outputRows=20,outputHeapSize=480 B,parti
tions=1)
     -> ProjectRestrict(n=4,totalCost=4.054,outputRows=20,outputHeapSize=480 B,parti
tions=1)
         -> Subquery(n=3,totalCost=12.55,outputRows=20,outputHeapSize=480 B,partition
s=1,correlated=true,expression=true,invariant=true)
             -> IndexScan[SYSTABLES_INDEX1(145)](n=2,totalCost=4.054,outputRows=20,outpu
tHeapSize=480 B,partitions=1,baseTable=SYSTABLES(48),preds=[(T2.TABLENAME[0:1] = T.T
ABLENAME[4:1])])
                 -> IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=4.054,outputRows=20,outputH
eapSize=480 B,partitions=1,baseTable=SYSTABLES(48))

6 rows selected
```

Nodes

- » Subqueries are listed as a second query tree, whose starting indentation level is the same as the `ProjectRestrict` operation that *owns* the subquery.
- » Includes a *correlated* field, which specifies whether or not the query is treated as correlated or uncorrelated.
- » Includes an *expression* field, which specifies whether or not the subquery is an expression.
- » Includes an *invariant* field, which indicates whether the subquery is invariant.

See Also

- » [About Explain Plan](#)

Splice Machine Logging

This topic describes the logging facility used in Splice Machine. Splice Machine also allows you to exercise direct control over logging in your database. You can:

- » [Manually disable logging](#) of all SQL statements.
- » Configure individual logger objects to log [specific message levels](#).
- » Call the `log` function for a specific logger object, passing it a logging level value and a message. If the logger is currently configured to record messages at the specified level, the message is added to the log; otherwise, the logger ignores the message. Logger levels are ordered hierarchically; any message with a level equal to or greater than the hierarchical level for which logging is enabled is recorded into the log.

Splice Machine uses the open source [Apache log4j Logging API](#), which allows you to associate a logger object with any java class, among other features. Loggers can be set to capture different levels of information.



Splice Machine enables statement logging by default.

Logging too much information can slow your system down, but not logging enough information makes it difficult to debug certain issues. The [Splice Machine Loggers](#) section below summarizes the loggers used by Splice Machine and the system components that they use. You can use the SQL logging functions described in the [SQL Logger Functions](#) section below to retrieve or modify the logging level of any logger used in Splice Machine.

Manually Disabling Logging

Logging of SQL statements is automatically enabled in Splice Machine; to disable logging of statements, you can do so in either of these ways:

- » You can pass an argument to your Splice Machine JVM startup script, as follows:

```
-Dderby.language.logStatementText=false
```

- » You can add the following property definition to your `hbase-default.xml`, `hbase-site.xml`, or `splice-site.xml` file:

```
<property>
<name>splice.debug.logStatementContext</name>
<value>false</value>
<description>Property to enable logging of all statements.</description>
</property>
```

You can examine the logged data in your region server's logs; if you want to change the location where events are logged, see the instructions in the Installation Guide for your platform (Cloudera, Hortonworks, or MapR).

Logger Levels

The Log4j API defines six logger levels. The following table displays them from the lowest level to the highest:

Logger Level	What gets logged for a logger object set to this level
TRACE	Captures all messages.
DEBUG	Captures any message whose level is DEBUG, INFO, WARN, ERROR, or FATAL.
INFO	Captures any message whose level is INFO, WARN, ERROR, or FATAL.
WARN	Captures any message whose level is WARN, ERROR, or FATAL.
ERROR	Captures any message whose level is ERROR or FATAL.
FATAL	Captures only messages whose level is FATAL.

Splice Machine Loggers

The following table summarizes the loggers used in the Splice Machine environment that might interest you if you're trying to debug performance issues in your database:

Logger Name	Default Level	Description
org.apache	ERROR	Logs all Apache software messages
com.splicemachine.db	WARN	Logs all Derby software messages
com.splicemachine.db.shared.common.sanity	ERROR	Logs all Derby Sanity Manager messages
com.splicemachine.derby.impl.sql.catalog	WARN	Logs Derby SQL catalog/ dictionary messages
com.splicemachine.db.impl.sql.execute.operations	WARN	Logs Derby SQL operation messages
org.apache.zookeeper.server.ZooKeeperServer	INFO	Used to determine when Zookeeper is started
org.apache.zookeeper.server.persistence.FileTxnSnapLog	INFO	Logs Zookeeper transactions

Logger Name	Default Level	Description
com.splicemachine	WARN	By default, controls all Splice Machine logging
com.splicemachine.derby.hbase.SpliceDriver	INFO	Prints start-up and shutdown messages to the log and to the console
com.splicemachine.derby.management.StatementManager	ERROR	Set the level of this logger to TRACE to record execution time for SQL statements

SQL Logger Functions

Splice Machine SQL includes the following built-in system procedures, all documented in our *SQL Reference Manual*, for interacting with the Splice Machine logs:

System Procedure	Description
SYSCS_UTIL.SYSCS_GET_LOGGERS	Displays a list of the active loggers.
SYSCS_UTIL.SYSCS_GET_LOGGER_LEVEL	Displays the current logging level of the specified logger.
SYSCS_UTIL.SYSCS_SET_LOGGER_LEVEL	Sets the current logging level of the specified logger.

See Also

- » [SYSCS_UTIL.SYSCS_GET_LOGGERS](#)
- » [SYSCS_UTIL.SYSCS_GET_LOGGER_LEVEL](#)
- » [SYSCS_UTIL.SYSCS_SET_LOGGER_LEVEL](#)

Debugging Splice Machine

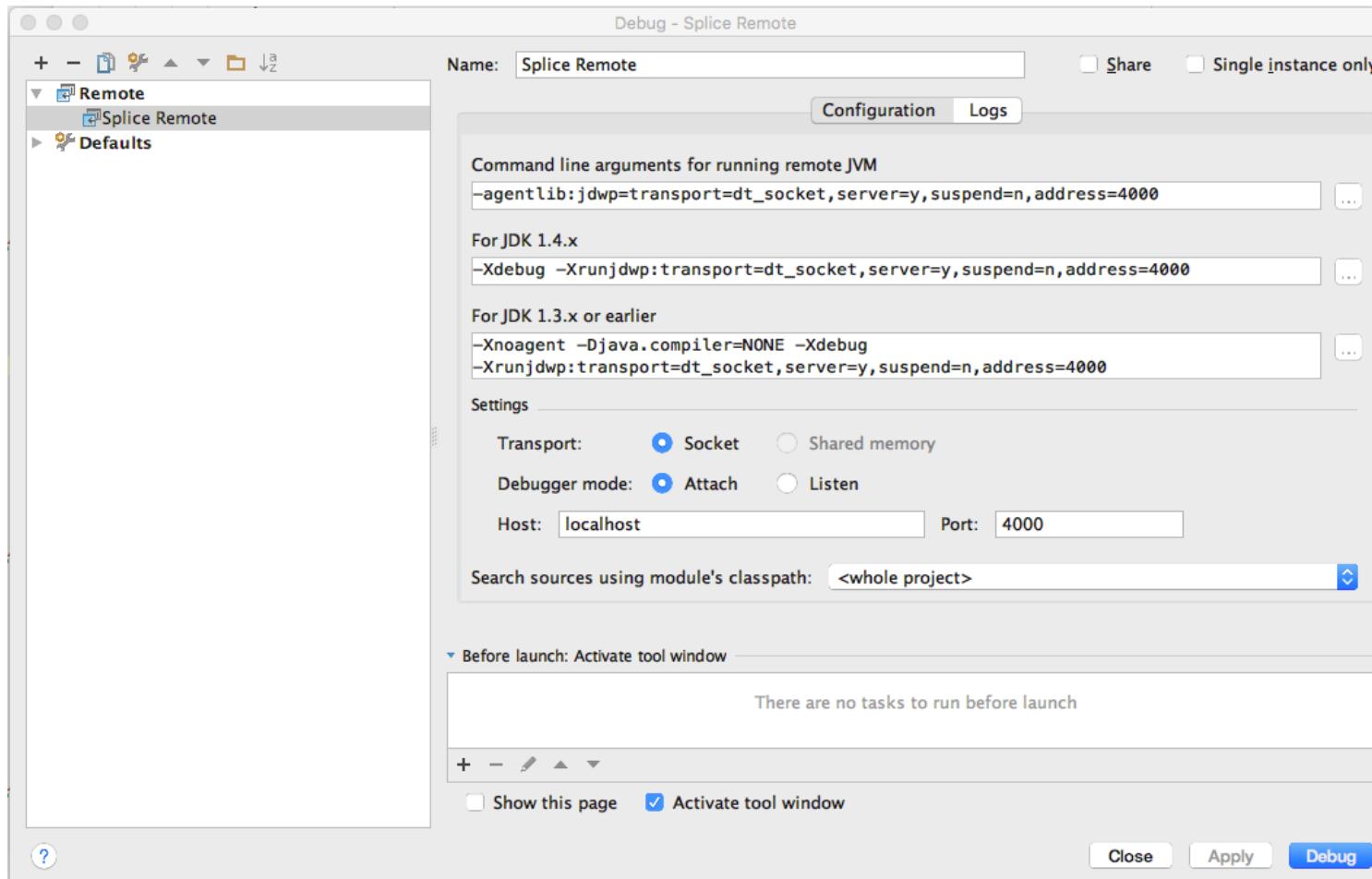
This topic describes the parameter values you need to know for debugging Splice Machine with a software tool:

- » Create a configuration in your software to remotely attach to Splice Machine
- » Connect to port 4000.

NOTE: If you're debugging code that is to be run in a Spark worker, connect to port 4020 instead.

Example

Here's an example of an *IntelliJ IDEA* debugging configuration using port 4000:





For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](#).

Using Splice Machine Snapshots

This topic describes how to use the Splice Machine snapshot feature to create a restorable snapshot of a table or schema; this is commonly used when importing or deleting a significant amount of data from a database.

Overview

Snapshots allow you to create point-in-time backups of tables (or an entire schema) without actually cloning the data.

NOTE: Snapshots include both the data and indexes for tables.

You use these system procedures and tables to work with snapshots:

- » Use the `SYSCS_UTIL.SNAPSHOT_TABLE` system procedure to create a named snapshot for a table.
- » Use the `SYSCS_UTIL.SNAPSHOT_SCHEMA` system procedure to create a named snapshot for a schema.
- » Use the `SYSCS_UTIL.RESTORE_SNAPSHOT` system procedure to restore a table or schema from a named snapshot.
- » Use the `SYSCS_UTIL.DELETE_SNAPSHOT` system procedure to delete a named snapshot.
- » Information about stored snapshots, including their names, is found in the `SYS.SYSSNAPSHOTS` system table.

Splice Machine Commands Reference

This section contains a reference topic page for each Splice Machine command. Another topic, Using the `splice>` Command Line Interface, presents general syntax and usage help for the `splice>` prompt.

- » [Commands you can use with all connections to Splice Machine databases](#), which means that you can use them with the `sqlshell` interface in our *On-Premise Database* and *Database-as-Service* products, and also with programs that connect to a Splice Machine database using JDBC or ODBC, including the Zeppelin notebook interface in our Database Service.
- » [Commands you can only use with the `splice> \(sqlshell.sh\)` command line interface in our *On-Premise Database* and *Database-as-Service* products](#). These are mostly commands that display information about the database in a terminal interface, and are not available through JDBC or ODBC connections.

Commands You Can Use with All Connections to a Splice Machine Database

The following table summarizes the commands that you can use them with the `sqlshell` interface in our *On-Premise Database* and *Database-as-Service* products, and also with programs that connect to a Splice Machine database using JDBC or ODBC, including the Zeppelin notebook interface in our Database Service.

Command	Description	Usage
Analyze	Collects statistics for a table or schema.	<code>splice> analyze table myTable;</code> <code>splice> analyze schema myschema;</code>
Autocommit	Turns the connection's auto-commit mode on or off.	<code>splice> autocommit off;</code>
Commit	Commits the currently active transaction and initiates a new transaction.	<code>splice> commit;</code>
Execute	Executes an SQL prepared statement or SQL command string.	<code>splice> execute 'insert into myTable(id, val) values(?,?)' ;</code>
Explain	Displays the execution plan for an SQL statement.	<code>splice> explain select count(*) from si;</code>
Export	Exports query results to CSV files.	<code>splice> EXPORT('/my/export/dir', null, null, null, null, null) SELECT a,b,sqrt(c) FROM join t2 on t1.a=t2.a;</code>
Prepare	Creates a prepared statement for use by other commands.	<code>splice> prepare seeMenu as 'SELECT * FROM menu';</code>

Command	Description	Usage
Release Savepoint	Releases a savepoint.	splice> release savepoint gSavePt1;
Remove	Removes a previously prepared statement.	splice> remove seeMenu;
Rollback	Rolls back the currently active transaction and initiates a new transaction.	splice> rollback;
Rollback to Savepoint	Rolls the current transaction back to the specified savepoint.	splice> rollback to savepoint gSavePt1;
Savepoint	Creates a savepoint within the current transaction.	splice> savepoint gSavePt1;

Commands You Can Only Use with Our Command Line Interface (`sqlshell.sh`)

The following table summarizes the commands that you can only use with the `splice>` command line interface (`sqlshell.sh`) in our *On-Premise Database* and *Database-as-Service* products.

Command	Description	Usage
Connect	Connect to a database via its URL.	splice> connect 'jdbc:splice://xyz:1527/ splicedb';
Describe	Displays a description of a table or view.	splice> describe myTable;
Disconnect	Disconnects from a database.	splice> disconnect SPICE;
Elapsedtime	Enables or disables display of elapsed time for command execution.	splice> elapsedtime on;
Exit	Causes the command line interface to exit.	splice> exit;
Help	Displays a list of the available commands.	splice> help;

Command	Description	Usage
MaximumDisplayWidth	Sets the maximum displayed width for each column of results displayed by the command line interpreter.	splice> maximumdisplaywidth 30;
Run	Runs commands from a file.	splice> run myCmdFile;
Set Connection	Allows you to specify which connection is the current connection	splice> set connection sample1;
Show Connections	Displays information about active connections and database objects.	splice> show connections;
Show Functions	Displays information about functions defined in the database or in a schema.	splice> show functions in splice;
Show Indexes	Displays information about the indexes defined on a table, a database, or a schema.	splice> show indexes from mytable;
Show Primary Keys	Displays information about the primary keys in a table.	splice> show primarykeys from mySchema.myTable;
Show Procedures	Displays information about active connections and database objects.	splice> show procedures in syscs_util;
Show Roles	Displays information about all of the roles defined in the database.	splice> show roles;
Show Schemas	Displays information about the schemas in the current connection.	splice> show schemas;
Show Synonyms	Displays information about the synonyms that have been created in a database or schema.	splice> show synonyms;
Show Tables	Displays information about all of the tables in a database or schema.	splice> show tables in SPLICE;
Show Views	Displays information about all of the active views in a schema.	splice> show views in SPLICE;

Using the splice> Command Line Interface

This topic presents information that will help you in using the Splice Machine `splice>` command line interpreter, in the following sections:

- » The [splice> Command Line Interpreter](#) section shows you how to invoke the `splice>` command line.
- » The [Command Line Output](#) section describes how you can adjust the appearance of output from the interepreter.
- » The [Command Line Syntax](#) section summarizes the syntax of commands, including capitalization and case-sensitivity rules, as well as various special characters you can use in your commands. It also shows you how to include comments in your command lines and how to run a file of SQL commands.
- » The [Example Command Lines](#) section shows several examples of command lines.
- » The Scripting Splice Commands tutorial describes how to create a script of `splice>` commands to run a series of operations like loading a number of files into your database

The remainder of this section contains a reference page for each of the command line commands.

splice> Command Line Interpreter

To run the Splice Machine command line interpreter, run the `sqlshell.sh` script in your terminal window.

```
% ./sqlshell.sh
splice>
```

When the interpreter prompts you with `splice>`, you can enter commands, ending each with a semicolon. For a complete description of `splice>` syntax, see the next section in this topic, [Command Line Syntax](#).

Note that you can optionally specify a file of sql commands that the interpreter will run using the `-f` parameter; after running those commands, the interpreter exits. For example:

```
./sqlshell.sh -f /home/mydir/sql/test.sql
```

You can optionally include parameter values when running `sqlshell.sh` script, to change default values. Here's the syntax:

```
sqlshell.sh [-h host] [-p port] [-u username] [-s password] [-f commandsFile]
```

-host

The hostname or IP address of your Splice Machine HBase RegionServer.

The default value is `localhost`.

-port

The port on which Splice Machine is listening for your connection.

The default value is 1527.

-username

The user name for your Splice Machine database.

The default value is `splice`.

-password

The password for your Splice Machine database.

The default value is `admin`.

-f [fileName]

The name of a file with SQL commands in it: `sqlshell` starts up the `splice>` command line interpreter, runs the commands in the file, and then exits. For example:

```
$ ./sqlshell.sh -f /home/mydir/sql/test.sql

===== rlwrap detected and enabled. Use up and down arrow keys to scroll through command line history. =====

Running Splice Machine SQL shell
For help: "splice> help;"
SPICE* -          jdbc:splice://10.1.1.111:1527/splicedb
* = current connection
splice> elapsedtime on;
splice> select count(*) from CUST_EMAIL;
1
-----
0

1 row selected
ELAPSED TIME = 6399 milliseconds
splice>
$
```

The `test.sql` file used in the above example contains the following commands:

```
elapsedtime on;
select count(*) from CUST_EMAIL;
```

Command Line Output

Output from `splice>` commands is displayed in your terminal window. The `maximumdisplaywidth` setting affects how the output is displayed; specifically, it determines if the content of each column is truncated to fit within the width that you specify.

When you set `maximumdisplaywidth` to 0, all output is displayed, without truncation.

Command Line Syntax

This section briefly summarizes the syntax of command lines you can enter in Zeppelin notebooks and in response to the `splice>` prompt, including these subsections:

- » [Finishing and submitting command lines](#)
- » [Capitalization and case sensitivity rules](#)
- » [Special character usage](#)
- » [Running multi-line commands](#)
- » [Running commands from a file](#)
- » [Including comments in your command lines](#)
- » [Using rlWrap on the command line](#)

Finish Commands with a Semicolon

The command line interface allows you to enter multi-line commands, and waits for a non-escaped semicolon (`;`) character to signal the end of the command.

A command is not executed until you enter the semicolon character and press the `Return` or `Enter` key.

Capitalization and Case Sensitivity Rules

Certain identifiers and keywords are case sensitive:

Identifier	Case Sensitive?	Notes and Example
SQL keywords	Not case sensitive	These are all equivalent: <code>SELECT</code> , <code>Select</code> , <code>select</code> , <code>SeLeCt</code> .
<i>ANSI SQL identifiers</i>	Not case sensitive	These are not case sensitive unless they are delimited.
<i>Java-style identifiers</i>	Always case sensitive	These are NOT equivalent: <code>my_name</code> , <code>My_Name</code> .

Special Characters You Can Use

The following table describes the special characters you can use in commands:

Purpose	Character(s) to use	Notes and example
<i>To delimit special identifiers</i>	Double quotation marks ("")	Special identifiers are also known as <i>delimited identifiers</i> .
<i>To delimit character strings</i>	Single quotation marks ('')	
<i>To escape a single quote or apostrophe within a character string</i>	Single quotation mark ('')	<p>Since single quotation marks are used to delimit strings, you must escape any single quotation marks you want included in the string itself.</p> <p>Use the single quotation mark itself as the escape character, which means that you enter two single quotation marks within a character string to include one single quotation mark.</p> <p>Example: 'This string includes "my quoted string" within it.'</p>
<i>To escape a double quote</i>	<i>Not needed</i>	You can simply include a double quotation mark in your command lines.
<i>To specify a wild card within a Select expression</i>	The asterisk (*) character	<p>This is the SQL metasymbol for selecting all matching entries.</p> <p>Example: SELECT * FROM MyTable;</p>
<i>To specify a wild card sequence in a string with the LIKE operator</i>	The percentage (%) character	Example: SELECT * FROM MyTable WHERE Name LIKE 'Ga%';
<i>To specify a single wild card character in a string with the LIKE operator</i>	The underline (_) character	Example: SELECT * FROM MyTable WHERE Name LIKE '%Er_n%';
<i>To begin a single-line comment</i>	Two dashes (--)	-- the following selects everything in my table: SELECT * FROM MyTable;
<i>To bracket a multi-line comment</i>	/* and */	<p>All text between the comment start /* and the comment end */ is ignored.</p> <pre>/* the following selects everything in my table, which we'll then display on the screen */ SELECT * FROM MyTable;</pre>

Entering Multi-line Commands

When using the command line (the `splice>` prompt), you must end each SQL statement with a semicolon (`;`). For example:

```
splice> select * from myTable;
```

You can extend SQL statements across multiple lines, as long as you end the last line with a semicolon. Note that the `splice>` command line interface prompts you with a fresh `>` at the beginning of each line. For example:

```
splice> select * from myTable> where i > 1;
```

Running SQL Statements From a File

You can also create a file that contains a collection of SQL statements, and then use the `run` command to run those statements. For example:

```
splice> run 'path/to/file.sql';
```

Including Comments

You can include comments on the command line or in a SQL statement file by prefacing the command with two dashes (`--`). Any text following the dashes is ignored by the SQL parser. For example:

```
splice> select * from myTable -- This selects everything in myTable;
```

Misaligned Quotes

If you mistakenly enter a command line that has misaligned quotation symbols, the interpreter can seem unresponsive. The solution is to add the missing quotation mark(s), followed by a semicolon, and resubmit the line. It won't work as expected, but it will enable you to keep working.

Using rlWrap on the Command Line

`rlWrap` is a Unix utility that Splice Machine encourages you to use: it allows you to scroll through your command line history, reuse and alter lines, and more. We've included a synopsis of it here.

Example Command Lines

Here are several example command lines:

Operation	Command Example
Display a list of all tables and their schemas	<code>splice> show tables;</code>

Operation	Command Example
Display the columns and attributes of a table	splice> describe tableA;
Limit the number of rows returned from a select statement	splice> select * from tableA { limit 10 };
Print a current time stamp	splice> values current_timestamp;

NOTE: Remember that you must end your command lines with the semicolon (;) character, which submits the command line to the interpreter.

Scripting splice> Commands

You can use the Splice Machine Command Line Interface (splice>) to interactively run database commands. This topic describes how to create a script of splice> commands to run a series of operations, such as loading a number of files into your database. To script a series of splice> commands, you need to create:

- » an SQL commands file containing the SQL statements you want executed
- » an SQL file to connect to the database and invoke the SQL commands file
- » a shell script using Bash (/bin/bash)

Follow these steps to create your script:

1. Create a file of SQL commands:

First, create a file that contains the SQL commands you want to run against your Splice Machine database. For this example, we'll create a file named `create-my-tables.sql` that creates a table in the database:

```
create table customers (
    CUSTOMER_ID BIGINT,
    FIRST_NAME VARCHAR(30),
    LAST_NAME VARCHAR(30)
);
```

2. Create an SQL file to connect to the database and invoke the commands file

We need a separate SQL file named `my_load_datascript.sql` that connects to your database and then invokes the file of SQL commands we just created.

NOTE: The `connect` command in this file must run before running the file of SQL statements.

Here we name the first SQL file, and define it to run the SQL statements file named `create-my-tables.sql`:

```
--First connect to the database
connect 'jdbc:splice://<regionServer>:1527/splicedb';

--Next run your sql file
run '/users/yourname/create-my-tables.sql';

show tables;
quit;
```

If you are running Splice Machine on a cluster, connect from a machine that is NOT running an HBase RegionServer and specify the IP address of a `regionServer` node, e.g. 10.1.1.110. If you're using the standalone version of Splice Machine, specify `localhost` instead.

3. Create a shell script to run your SQL connect file

We now create a shell script named `load_datascript.sh` to run the `my_load_datascript.sql` file:

```
#!/bin/bash

export CLASSPATH=<FULL_PATH_TO_SPLICEMACHINE_JAR_FILE>
java -Djdbc.drivers=com.splicemachine.db.jdbc.ClientDriver -Dij.outfile=m
y_load_datascript.out com.splicemachine.db.tools.ij < my_load_datascrip
t.sql
```

The first line of this script must set the `CLASSPATH` to the location of your Splice Machine jar file. The second line runs the `ij` command, specifying its output file (`my_load_datascript.out`) and its SQL commands input file, which is the `my_load_datascript_sql` file that we created in the previous step.

4. Make your shell script executable

We need to make the shell script executable with the `chmod` shell command:

```
chmod +x load_datascript.sh
```

5. Use nohup to run the script

The `nohup` utility allows you to run a script file in the background, which means that it will continue running if you log out, disconnect from a remote machine, or lose your network connection.

```
nohup ./load_datascript.sh > ./load_datascript.out 2>&1 &
```

Here's the syntax for the `nohup` utility:

```
nohup ./command-name.sh > ./command-name.out 2>&1 &
```

command-name.sh

The name of the shell script or a command name.

command-name.out

The name of the file to capture any output written to `stdout`.

2>&1

This causes `stderr` (file descriptor 2) to be written to `stdout` (file descriptor 1); this means that all output will be captured in `command-name.out`.

&

The `nohup` utility does not automatically run its command in the background, so we add the `&` to do

Analyze Command

The analyze command collects statistics for a specific table, or for an entire schema.

NOTE: Once statistics have been collected for a schema or table, they are automatically used by the query optimizer.

Syntax

```
ANALYZE TABLE [schemaName '.' ] table-Name  
[ESTIMATE STATISTICS SAMPLE samplepercent PERCENT];  
ANALYZE SCHEMA schema-Name;
```

table-Name

The name of the table you want to analyze, which can optionally be qualified by its schema name. If you don't specify a *schemaName*, the current schema is assumed.

You must have insert permission for the table to be able to run this command.

schema-Name

The name of the schema you want to analyze.

You must have insert permission for all tables in the schema to be able to run this command.

samplepercent

A value between 0 and 100 that specifies the sampling percentage to use when generating statistics for this table.

If you include this clause, statistics are generated by sampling the specified sampling percentage of the table. This can significantly reduce the overhead associated with generating statistics.

If you do not include this clause, statistics are generated based on the full table.

Analyze Table

The ANALYZE TABLE command collects statistics for a specific table in the current schema. It also collects statistics for the index associated with the table in the schema. For example, if you have the following in your database:

- » a table named myTable
- » myTable has two indices: myTableIndex1 and myTableIndex2

Then ANALYZE TABLE will collect statistics for myTable, myTableIndex1, and myTableIndex2.

The ANALYZE TABLE command displays the following information for each partition of the table:

Value	Description								
schemaName	The name of the schema.								
tableName	The name of the table.								
partition	The Splice Machine partition. We merge the statistics for all table partitions, so the partition will show as –All– when you specify one of the non-merged type values for the statsType parameter.								
rowsCollected	The total number of rows collected for the table.								
partitionSize	The combined size of the table's partitions.								
statsType	<p>The type of statistics, which is one of these values:</p> <table border="1" style="margin-left: 20px;"> <tr> <td>0</td><td>Full table (not sampled) statistics that reflect the unmerged partition values.</td></tr> <tr> <td>1</td><td>Sampled statistics that reflect the unmerged partition values.</td></tr> <tr> <td>2</td><td>Full table (not sampled) statistics that reflect the table values after all partitions have been merged.</td></tr> <tr> <td>3</td><td>Sampled statistics that reflect the table values after all partitions have been merged.</td></tr> </table>	0	Full table (not sampled) statistics that reflect the unmerged partition values.	1	Sampled statistics that reflect the unmerged partition values.	2	Full table (not sampled) statistics that reflect the table values after all partitions have been merged.	3	Sampled statistics that reflect the table values after all partitions have been merged.
0	Full table (not sampled) statistics that reflect the unmerged partition values.								
1	Sampled statistics that reflect the unmerged partition values.								
2	Full table (not sampled) statistics that reflect the table values after all partitions have been merged.								
3	Sampled statistics that reflect the table values after all partitions have been merged.								
sampleFraction	<p>The sampling percentage, expressed as 0 . 0 to 1 . 0,</p> <ul style="list-style-type: none"> » If statsType=0 or statsType=1 (full statistics), this value is not used, and is shown as 0. » If statsType=2 or statsType=3, this value is the percentage or rows to be sampled. A value of 0 means no rows, and a value of 1 means all rows (same as full statistics). 								

Analyze Schema

The ANALYZE SCHEMA command collects statistics for every table in the schema. It also collects statistics for the index associated with every table in the schema. For example, if you have the following situation:

- » a schema named mySchema
- » mySchema contains two tables: myTable1 and myTable2
- » myTable1 has two indices: myTable1Index1 and myTable1Index2

Then the ANALYZE SCHEMA command will collect statistics for myTable1, myTable2, myTable1Index1, and myTable1Index2.

The ANALYZE SCHEMA command displays the same information as shown for ANALYZE TABLE, for each table in the schema.

NOTE: This command operates like the `SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS` built-in system procedure.

Examples

```
splice> analyze table test.t2;
schemaName |tableName |partition |rowsCollec&|partitionSize |partitionCount |statsT
ype |sampleFraction
-----
-----  

TEST      |T2        |-All-       |39226        |235356          |1
|2          |0  

1 rows selected
splice>analyze table test.t2 estimate statistics sample 50 percent;
schemaName |tableName |partition |rowsCollec&|partitionSize |partitionCount |statsT
ype |sampleFraction
-----
-----  

TEST      |T2        |-All-       |19613        |235356          |1
|3          |0.5  

1 rows selected
splice>analyze schema test;
schemaName |tableName |partition |rowsCollec&|partitionSize |partitionCount |statsT
ype |sampleFraction
-----
-----  

TEST      |T2        |-All-       |39226        |235356          |1
|2          |0
TEST      |T5        |-All-       |39226        |235356          |1
|2          |0
2 rows selected
splice>
```

Autocommit Command

The `autocommit` command enables or disables auto-commit mode.

JDBC specifies that the default auto-commit mode is enabled; however, certain types of processing require that auto-commit mode be disabled.

Syntax

```
AUTOCOMMIT {ON | OFF}
```

ON

Enables auto-commit mode.

If auto-commit mode is changed from disabled (`OFF`) to enabled (`on`) when there is a transaction outstanding, that work is committed when the current transaction commits, not at the time auto-commit is enabled. Thus, if you are enabling auto-commit when a transaction is outstanding, first use either the `Rollback` command to ensure that all prior work is completed before the return to auto-commit mode.

OFF

Disables auto-commit mode.

Examples

```
splice> autocommit off;
splice> DROP TABLE menu;
0 rows inserted/updated/deleted
splice> CREATE TABLE menu (course CHAR(10), item CHAR(20), price INT);
0 rows inserted/updated/deleted
splice> INSERT INTO menu VALUES ('entree', 'lamb chop', 14),
('dessert', 'creme brulee', 6),
('appetizer', 'baby greens', 7);
3 rows inserted/updated/deleted
splice> commit;
splice> autocommit on;
splice>
```

Commit Command

The `commit` command issues a `java.sql.Connection.commit` request, which commits the currently active transaction and initiates a new transaction.

NOTE: You should only use this command when auto-commit mode is disabled.

Syntax

```
COMMIT
```

Examples

```
splice> commit;  
splice>
```

Execute Command

The execute command executes an SQL command string or a prepared statement.

Syntax

```
EXECUTE { SQLString | PreparedStatementIdentifier }
          [ USING { String | Identifier } ]
```

SQLString

The SQL command string to execute; this string is passed to the connection without further processing by the command line interpreter.

PreparedStatementIdentifier

The identifier of the prepared statement to execute; this must be the name associated with a prepared statement when created by the Prepare command.

String

Use this or *Identifier* to supply values for dynamic parameters, if the command being executed contains them.

Identifier

Use this or *String* to supply values for dynamic parameters, if the command being executed contains them. This identifier must have a result set as its result:

- » Each row of the result set is applied to the input parameters of the command to be executed, so the number of columns in the Using clause's result set must match the number of input parameters in the statement being executed.
- » The command line interpreter displays the results of each execution of the statement as they are created.
- » If the Using clause's result set contains zero rows, the statement is not executed.

Examples

```

splice> autocommit off;
splice> prepare menuInsert as 'INSERT INTO menu VALUES (?, ?, ?)';
splice> execute menuInsert using 'VALUES
(''entree'', ''lamb chop'', 14),
(''dessert'', ''creme brulee'', 6)';
1 row inserted/updated/deleted
1 row inserted/updated/deleted
splice> commit;
splice> connect 'jdbc:splice://abc:1527/splicedb;user=me;password=mypswd';
splice> create table firsttable (id int primary key,
name varchar(12));
0 rows inserted/updated/deleted
splice> insert into firsttable values
10,'TEN'),(20,'TWENTY'),(30,'THIRTY');
3 rows inserted/updated/deleted
splice> select * from firsttable;
ID      | NAME
-----
10     | TEN
20     | TWENTY
30     | THIRTY

3 rows selected
splice> connect 'jdbc:splice://xyz:1527/spicedb';
splice(CONNECTION1)> create table newtable (newid int primary key,
newname varchar(12));
0 rows inserted/updated/deleted
splice(CONNECTION1)> prepare src@connection0 as 'select * from firsttable';
splice(CONNECTION1)> autocommit off;
splice(CONNECTION1)> execute 'insert into newtable(newid, newname)
values(?,?)' using src@connection0;
1 row inserted/updated/deleted
1 row inserted/updated/deleted
1 row inserted/updated/deleted
splice(CONNECTION1)> commit;
splice(CONNECTION1)> select * from newtable;
NEWID      | NEWNAME
-----
10     | TEN
20     | TWENTY
30     | THIRTY

3 rows selected
splice(CONNECTION1)> show connections;
CONNECTION0 -  jdbc:splice://abc:1527/spicedb
CONNECTION1* -  jdbc:splice://xyz:1527/spicedb
splice(CONNECTION1)> disconnect connection0;
splice>
```

Explain Plan Command

The `explain` command displays the execution plan for a statement without actually executing the statement; it parses and optimizes the SQL, then presents its execution plan.

You can use this to tune a query for improved performance.

Syntax

```
explain Statement
```

Statement

An SQL statement.

Usage

SQL Data Definition Language (DDL) statements have no known cost, and thus do not require optimization. Because of this, the `explain` command does not work with DDL statements; attempting to `explain` a DDL statement such as `CREATE TABLE` will generate a syntax error.

For more information about using the `explain` command, including a number of annotated examples, see [Explain Plan](#).

Examples

```
splice> explain select * from t t1 where t1.i < (select max(i) from t t1);
```

The [Explain Plan](#) topic contains a number of examples that are described in detail.

Export Command

The `export` command exports the results of an SQL query to a CSV (comma separated value) file.

Syntax

```
EXPORT ( exportPath,
        compression,
        replicationCount,
        fileEncoding,
        fieldSeparator,
        quoteCharacter ) <SQL_QUERY>;
```

exportPath

The directory in which you want the export file(s) written.

compress

Whether or not to compress the exported files. You can specify one of the following values:

Value	Description
<code>true</code>	The exported files are compressed using deflate/gzip.
<code>false</code>	Exported files are not compressed.

replicationCount

The file system block replication count to use for the exported CSV files.

You can specify any positive integer value. The default value is 1.

fileEncoding

The character set encoding to use for the exported CSV files.

You can specify any character set encoding that is supported by the Java Virtual Machine (JVM). The default encoding is UTF-8.

fieldSeparator

The character to use for separating fields in the exported CSV files.

The default separator character is the comma (,).

quoteCharacter

The character to use for quoting output in the exported CSV files.

The default quote character is the double quotation mark (").

Usage

The EXPORT command generates one or more CSV files and stores them in the directory that you specified in the `exportPath` parameter. More than one output file is generated to enhance the parallelism and performance of this operation.

If `compression=true`, then each of the generated files is named with this format:

```
export_<N>.csv.gz
```

If `compression=false`, then each of the generated files is named with this format:

```
export_<N>.csv
```

The value of `<N>` is a random integer value.

Merging the Exported Files

You can copy all of the exported files into a single file on your local file system using the Hadoop FS command `getmerge`. The syntax for `getmerge` is:

```
hadoop fs -getmerge sourceDir localPath
```

Use the `exportPath` directory as the value of `sourceDir` to copy all of the exported CSV files to your `localPath`.

For more information about the `getmerge` command, see <http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html#getmerge>.

Examples

```
-- This example uses all default options:  
splice> EXPORT('/my/export/dir', false, null, null, null, null)  
      SELECT a,b,sqrt(c) FROM t1 join t2 on t1.a=t2.a;  
  
-- This example explicitly specifies options:  
splice> EXPORT('/my/export/dir', false, 3, 'utf-8', '|', ';')  
      SELECT a,b,sqrt(c) FROM t1 join t2 on t1.a=t2.a;
```

Prepare Command

The `prepare` command creates a `java.sql.PreparedStatement` using the value of the specified SQL command `String`, and assigns an identifier to the prepared statement so that other `splice>` commands can use the statement.

If a prepared statement with the specified *Identifier* name already exists in the command interpreter, an error is returned, and the previous prepared statement is left unchanged. If there are any errors in preparing the statement, no prepared statement is created.

If the *Identifier* specifies a connection Name, the statement is prepared on the specified connection.

Syntax

```
PREPARE Identifier AS String
```

Identifier

The identifier to assign to the prepared statement.

String

The command string to prepare.

Examples

```
splice> prepare seeMenu as 'SELECT * FROM menu';
splice> execute seeMenu;
COURSE      | ITEM          | PRICE
-----
entree     | lamb chop        | 14
dessert    | creme brulee     | 6

splice> prepare addYears as 'update children set age = age + ? where name = ?';
splice> execute addYears using 'values (10, ''Abigail'')';
```

Release Savepoint Command

The `release savepoint` command issues a `java.sql.Connection.releaseSavepoint` request, which releases a savepoint within the current transaction. Once a savepoint has been released, attempting to reference it in a rollback operation will cause an `SQLException` to be thrown.

When you commit a transaction, any savepoints created in that transaction are automatically released and invalidated when the transaction is committed or the entire transaction is rolled back.

NOTE: When you rollback a transaction to a savepoint, that savepoint and any others created after it within the transaction are automatically released.

Syntax

```
release savepoint identifier;
```

identifier

The name of the savepoint to release.

Examples

Example

First we'll create a table, turn `autocommit` off, and insert some data into the table. We then create a savepoint, and verify the contents of our table:

```
splice> CREATE TABLE myTbl(i int);
0 rows inserted/updated/deleted
splice> AUTOCOMMIT OFF;
splice> INSERT INTO myTbl VALUES 1,2,3;
3 rows inserted/updated/deleted
splice> SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3
3 rows selected
```

Next we add new values to the table and again verify its contents:

```
splice> INSERT INTO myTbl VALUES 4,5;
2 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3
4
5

5 rows selected
```

Now we release our original savepoint, insert a few more values, and create a new savepoint, savept2.

```
splice> RELEASE SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> INSERT INTO myTbl VALUES 6,7;
2 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3
4
5
6
7

7 rows selected
splice> SAVEPOINT savept2;
0 rows inserted/updated/deleted
```

We again insert data into the table, display its contents, and then do a rollback:

```
splice> INSERT INTO myTbl VALUES 8,9;
2 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3
4
5
6
7
8
9

9 rows selected
splice> ROLLBACK TO SAVEPOINT savept1;
ERROR 3B001: Savepoint SAVEPT1 does not exist or is not active in the current transaction.
splice> ROLLBACK TO SAVEPOINT savept2;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3
4
5
6
7

7 rows selected
```

And finally, we commit the transaction:

```
COMMIT;
```

See Also

- » [savepoint command](#)
- » [rollback to savepoint command](#)
- » The [*Running Transactions*](#) topic contains includes a discussion of using savepoints.

Remove Command

The `remove` command removes a previously prepared statement from the command line interpreter.

The statement is closed, releasing its database resources.

Syntax

```
REMOVE Identifier
```

Identifier

The name assigned to the prepared statement when it was prepared with the `Prepare` statement.

Examples

```
splice> prepare seeMenu as 'SELECT * FROM menu';
splice> execute seeMenu;
COURSE      | ITEM              | PRICE
-----
entree     | lamb chop           | 14
dessert    | creme brulee        | 6

2 rows selected
splice> remove seeMenu;
splice> execute seeMenu;
splice ERROR: Unable to establish prepared statement SEEMENU
splice>
```

Rollback Command

The `rollback` command issues a `java.sql.Connection.rollback` request, which rolls back (undoes) the currently active transaction and initiates a new transaction.

NOTE: You should only use this command when auto-commit mode is disabled.

Usage Notes

In contrast to the [Rollback to Savepoint](#) command, the `Rollback` command aborts the current transaction and starts a new one.

Examples

```
splice> autocommit off;
splice> INSERT INTO menu VALUES ('dessert', 'rhubarb pie', 4);
1 row inserted/updated/deleted
splice> SELECT * from menu;
COURSE | ITEM | PRICE
-----
entree | lamb chop | 14
dessert | creme brulee | 7
appetizer | baby greens | 7
dessert | rhubarb pie | 4

4 rows selected
splice> rollback;
splice> SELECT * FROM menu;
COURSE | ITEM | PRICE
-----
entree | lamb chop | 14
dessert | creme brulee | 7
appetizer | baby greens | 7

3 rows selected
splice>
```

Rollback to Savepoint Command

The `rollback to savepoint` command issues a `java.sql.Connection.rollback` request, which has been overloaded to work with a savepoint within the current transaction.

NOTE: When you rollback a transaction to a savepoint, that savepoint and any others created after it within the transaction are automatically released.

Syntax

```
rollback to savepoint identifier;
```

identifier

The name of the savepoint to which the transaction should be rolled back: all savepoints up to and including this one are rolled back.

Usage Notes

In contrast to the [Rollback](#) command, the `Rollback to Savepoint` command rolls back *but* of your work, but does not start a new transaction.

Examples

First we'll create a table, turn autocommit off, and insert some data into the table. We then create a savepoint, and verify the contents of our table:

```
splice> CREATE TABLE myTbl(i int);
0 rows inserted/updated/deleted
splice> AUTOCOMMIT OFF;
splice> INSERT INTO myTbl VALUES 1,2,3;
3 rows inserted/updated/deleted
splice> SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3
3 rows selected
```

Next we add new values to the table and again verify its contents:

```
splice> INSERT INTO myTbl VALUES 4,5;
2 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3
4
5
5 rows selected
```

Now we roll back to our savepoint, and verify that the rollback worked:

```
splice> ROLLBACK TO SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3
3 rows selected
```

And finally, we commit the transaction:

```
COMMIT;
```

See Also

- » [savepoint command](#)
- » [release savepoint command](#)
- » The [*Running Transactions*](#) topic contains includes a discussion of using savepoints.

Savepoint Command

The savepoint command issues a `java.sql.Connection.setSavepoint` request, which sets a savepoint within the current transaction.

Savepoints are only useful when autocommit is off.

NOTE: You can define multiple savepoints within a transaction.

Syntax

```
savepoint identifier;
```

identifier

An identifier name for the string.

Example

Example

First we'll create a table, turn autocommit off, and insert some data into the table. We then create a savepoint, and verify the contents of our table:

```
splice> CREATE TABLE myTbl(i int);
0 rows inserted/updated/deleted
splice> AUTOCOMMIT OFF;
splice> INSERT INTO myTbl VALUES 1,2,3;
3 rows inserted/updated/deleted
splice> SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3
3 rows selected
```

Next we add new values to the table and again verify its contents:

```
splice> INSERT INTO myTbl VALUES 4,5;
2 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3
4
5

5 rows selected
```

Now we roll back to our savepoint, and verify that the rollback worked:

```
splice> ROLLBACK TO SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----
1
2
3

3 rows selected
```

And finally, we commit the transaction:

```
COMMIT;
```

See Also

- » [release savepoint command](#)
- » [rollback to savepoint command](#)
- » The [Running Transactions](#) topic contains includes a discussion of using savepoints.

Connect Command

The connect command connects to the database specified by `ConnectionURLString`. It connects by issuing a `getConnection` request with the specified URL, using `java.sql.DriverManager` or `javax.sql.DataSource` to set the current connection to that URL.

Syntax

```
CONNECT ConnectionURLString [ AS Identifier ]
```

ConnectionURLString

The URL of the database. Note that this URL typically includes connection parameters such as user name, password, or security options.

Identifier

The optional name that you want to assign to the connection.

If the connection succeeds, the connection becomes the current one and all further commands are processed against the new, current connection.

Example 1: Connecting on a Cluster

If you are running Splice Machine on a cluster, connect from a machine that is NOT running an HBase RegionServer and specify the IP address of a `regionServer` node, e.g. 10.1.1.110.

```
splice> connect 'jdbc:splice://regionServer:1527/splicedb';
```

This example includes a user ID and password in the connection URL string:

```
splice> connect 'jdbc:splice://1.2.3.456:1527/spicedb;user=splice;password=admin';
```

And this example includes specifies that SSL peer authentication will be enabled for the connection:

```
splice> connect 'jdbc:splice://1.2.3.456:1527/spicedb;user=splice;password=admin;ssl=peerAuthentication';
```

NOTE: You can only connect with SSL/TLS if your administrator has configured this capability for you.

Example 2: Connecting to the Standalone Version

If you're using the standalone version of Splice Machine, specify `localhost` instead.

```
splice> connect 'jdbc:splice://localhost:1527/splicedb';
```

Here is an example that includes a user ID and password in the connect string:

```
splice> connect 'jdbc:splice://localhost:1527/splicedb;user=joey;password=bosssman';
```

Describe Command

The describe command displays a description of the specified table or view.

Syntax

```
DESCRIBE { table-Name | view-Name }
```

tableName

The name of the table whose description you want to see.

viewName

The name of the view whose description you want to see.

Results

The describe command results contains the following columns:

Column Name	Description
COLUMN_NAME	The name of the column
TYPE_NAME	The data type of the column
DECIMAL_DIGITS	The number of fractional digits
NUM_PREC_RADIX	The radix, which is typically either 10 or 2
COLUMN_SIZE	The column size: <ul style="list-style-type: none"> » For char or date types, this is the maximum number of characters » For numeric or decimal types, this is the precision
COLUMN_DEF	The default value for the column
CHAR_OCTE	Maximum number of bytes in the column
IS_NULL	Whether (YES) or not (NO) the column can contain null values

Examples

```
splice> describe T_DETAIL;
COLUMN_NAME          | TYPE_NAME | DEC | NUM | COLUM | COLUMN_DEF | CHAR_OCTE | IS_NUL
L
-----
TRANSACTION_HEADER_KEY | BIGINT    | 0   | 10  | 19   | NULL      | NULL      | NO
TRANSACTION_DETAIL_KEY | BIGINT    | 0   | 10  | 19   | NULL      | NULL      | NO
CUSTOMER_MASTER_ID    | BIGINT    | 0   | 10  | 19   | NULL      | NULL      | YES
TRANSACTION_DT         | DATE      | 0   | 10  | 10   | NULL      | NULL      | NO
ORIGINAL_SKU_CATEGORY_ID | INTEGER  | 0   | 10  | 10   | NULL      | NULL      | YES
5 rows selected
```

Disconnect Command

The disconnect command disconnects from a database. It issues a `java.sql.Connection.close` request for the current connection, or for the connection(s) specified on the command line.

Note that disconnecting from a database does not stop the command line interface or shut down Splice Machine. You can use the exit command to close out of the command line interface.

Syntax

```
DISCONNECT [ALL | CURRENT | connectionIdentifier]
```

ALL

All known connections are closed; as a result, there will not be a current connection.

CURRENT

The current connection is closed. This is the default behavior.

connectionIdentifier

The name of the connection to close; this must be same identifier assigned when the connection was opened with a Connect command.

Examples

```
splice> connect 'jdbc:splice://xyz:1527/splicedb';
splice> -- we create a new table in splicedb:
CREATE TABLE menu(course CHAR(10), ITEM char(20), PRICE integer);
0 rows inserted/updated/deleted
splice> disconnect;
splice>
```

ElapsedTime Command

The `elapsedtime` command enables or disables having the command line interface display the amount of time required for a command to complete its execution.

Syntax

```
ELAPSEDTIME { ON | OFF }
```

ON

Enables display of the elapsed time by the command line interface. When this is enabled, you'll see how much time elapsed during execution of the command line.

OFF

Disables display of the elapsed time.

Examples

```
splice> elapsedtime on;
splice> VALUES current_date;
1
-----
2014-06-24
ELAPSED TIME = 2134 milliseconds
splice>
```

Exit Command

The `exit` or `quit` command causes the command line interface to exit. Issuing this command from within a command file causes the outermost input loop to halt.

Syntax

```
EXIT
```

Examples

```
splice> exit;
```

Help Command

The `help` command displays a list of the available `splice>` commands.

Syntax

```
HELP
```

Example

```
splice> help;
```

Supported commands include:

```
CONNECT 'url for database' [ PROTOCOL namedProtocol ] [ AS connectionName ];
          -- connects to database URL
          -- and may assign identifier
AUTOCOMMIT [ ON | OFF ];           -- sets autocommit mode for the connection
SHOW SCHEMAS;                     -- lists all schemas in the current database
SHOW [ TABLES | VIEWS | PROCEDURES | FUNCTIONS | SYNONYMS ] { IN schema };
          -- lists tables, views, procedures, functions or synonyms
SHOW INDEXES { IN schema | FROM table };
          -- lists indexes in a schema, or for a table
SHOW ROLES;                       -- lists all defined roles in the database,
          -- sorted
SHOW SETTABLE_ROLES;              -- lists the roles which can be set for the
          -- current connection, sorted
DESCRIBE name;                   -- lists columns in the named table
COMMIT;                            -- commits the current transaction
ROLLBACK;                          -- rolls back the current transaction
PREPARE name AS 'SQL text';      -- prepares the SQL text
EXECUTE { name | 'SQL text' } [ USING { name | 'SQL text' } ] ;
          -- executes the statement with parameter
          -- values from the USING result set row
REMOVE name;                      -- removes the named previously prepared statement
RUN 'filename';                  -- run commands from the named file
ELAPSEDTIME [ ON | OFF ];        -- sets elapsed time mode for splice
MAXIMUMDISPLAYWIDTH integerValue;
          -- sets the maximum display width for
          -- each column to integerValue
EXIT;                             -- exits splice
HELP;                            -- shows this message
```

Any unrecognized commands are treated as potential SQL commands and executed directly.

```
splice>
```

MaximumDisplayWidth Command

The `maximumdisplaywidth` command sets the largest display width, in characters, for displayed columns in the command line interpreter.

This is generally used to increase the default value in order to display large blocks of text.

Syntax

```
MAXIMUMDISPLAYWIDTH integer_value
```

integer_value

The maximum width of each column that is displayed by the command line interpreter.

Set this value to 0 to display the entire content of each column.

Examples

```
splice> maximumdisplaywidth 3;
splice> VALUES 'NOW IS THE TIME!';
1
---
NOW
splice> maximumdisplaywidth 30;
splice> VALUES 'NOW IS THE TIME!';
1
-----
NOW IS THE TIME!
```

Run Command

The `run` command redirects the command line interpreter to read and process commands from the specified file. This continues until the end of the file is reached, or an exit command is executed. Note that the file *can* contain `run` commands.

NOTE: You can specify a file that is in the directory where you are running the command, or you can include the full file path so that the `run` command can find it.

The command line interpreter prints out the statements in the file as it executes them.

Syntax

```
RUN String
```

String

The name of the file containing commands to execute.

Examples

```
splice> run 'setupMenuConn.spl';
splice> -- this is setupMenuConn.spl
-- splice displays its contents as it processes file
splice> connect 'jdbc:splice://xyz:1527/splicedb';
splice> autocommit off;
splice> -- this is the end of setupMenuConn.spl
-- there is now a connection to splicedb on xyz and no autocommit.
-- input will now resume from the previous source.
;
splice>
```

Set Connection Command

The `set connection` command allows you to specify which connection is the current connection, when you have more than one connection open.

NOTE: If the specified connection does not exist, an error results, and the current connection is unchanged.

Syntax

```
SET CONNECTION Identifier
```

Identifier

The name of the connection that you want to be the current connection.

Examples

```
splice> connect 'jdbc:splice://abc:1527/splicedb;user=splice;password=admin' as sample1;
splice> connect 'jdbc:splice://xyz:1527/splicedb' as sample2;
splice (NEWDB)> show connections;
SAMPLE1 -    jdbc:splice://abc:1527/splicedb
SAMPLE2* -   jdbc:splice://xyz:1527/spicedb
* = current connection
splice(SAMPLE2)> set connection sample1;
splice(SAMPLE1)> disconnect all;
splice>
```

Show Connections

The `show connections` command displays a list of connection names and the URLs used to connect to them. The name of the current connection is marked with a trailing asterisk (*).

Syntax

```
SHOW CONNECTIONS
```

Examples

```
splice> connect 'jdbc:splice://abc:1527/splicedb' as sample1;
splice> connect 'jdbc:splice://xyz:1527/splicedb' as sample2;
splice (NEWDB)> show connections;
SAMPLE1 -    jdbc:splice://abc:1527/splicedb
SAMPLE2* -   jdbc:splice://xyz:1527/spicedb
* = current connection
splice(NEWDB)>
```

Show Functions

The `show functions` command displays all of the functions in the database, or the names of the functions in the specified schema.

Syntax

```
SHOW FUNCTIONS [ IN schemaName ]
```

schemaName

If you supply a schema name, only the functions in that schema are displayed; otherwise, all functions in the database are displayed.

Results

The `show functions` command results contains the following columns:

Column Name	Description
FUNCTION_SCHEMA	The name of the function's schema
FUNCTION_NAME	The name of the function
REMARKS	Any remarks that have been stored for the function

Examples

```
splice> CREATE FUNCTION TO_DEGREES ( RADIANS DOUBLE )
> RETURNS DOUBLE
> PARAMETER STYLE JAVA
> NO SQL LANGUAGE JAVA
> EXTERNAL NAME 'java.lang.Math.toDegrees';
0 rows inserted/updated/deleted
splice> show functions in splice;
FUNCTION_SCHEMA|FUNCTION_NAME | REMARKS
-----
SPICE          | TO_DEGREES           | java.lang.Math.toDegrees
1 row selected
```

Show Indexes

The `show indexes` command displays all of the indexes in the database, the indexes in the specified schema, or the indexes on the specified table.

Syntax

```
SHOW INDEXES [ IN schemaName | FROM tableName ]
```

schemaName

If you supply a schema name, only the indexes in that schema are displayed.

tableName

If you supply a table name, only the indexes on that table are displayed.

Results

The `show indexes` command results contains the following columns:

Column Name	Description
TABLE_NAME	The name of the table.
INDEX_NAME	The name of the index.
COLUMN_NAME	The name of the column.
ORDINAL	The position of the column in the index.
NON_UNIQUE	Whether this is a unique or non-unique index.
TYPE	The index type
ASC_	Indicates if this is an ascending (A) or descending (D) index.
CONGLOM_NO	The conglomerate number, which points to the corresponding table in HBase.

Examples

```
splice> show indexes from my_table;
TABLE_NAME      | INDEX_NAME          | COLUMN_NAME | ORDINAL& | NON_UNIQUE | TYPE  | ASC& | CONGL
OM_NO
-----
MY_TABLE        | I1                  | ID          | 1          | true      | BTREE | A    | 1937
MY_TABLE        | I1                  | STATE_CD    | 2          | true      | BTREE | A    | 1937
MY_TABLE        | I1                  | CITY        | 3          | true      | BTREE | A    | 1937
MY_TABLE        | I2                  | ID          | 1          | true      | BTREE | D    | 1953
MY_TABLE        | I2                  | STATE_CD    | 2          | true      | BTREE | D    | 1953
MY_TABLE        | I2                  | CITY        | 3          | true      | BTREE | D    | 1953
MY_TABLE        | I3                  | ID          | 1          | true      | BTREE | D    | 1969
MY_TABLE        | I3                  | STATE_CD    | 2          | true      | BTREE | A    | 1969
MY_TABLE        | I3                  | CITY        | 3          | true      | BTREE | D    | 1969
MY_TABLE        | I4                  | LATITUDE    | 1          | true      | BTREE | D    | 1985
MY_TABLE        | I4                  | STATE_CD    | 2          | true      | BTREE | A    | 1985
MY_TABLE        | I4                  | ID          | 3          | true      | BTREE | A    | 1985
MY_TABLE        | I5                  | ID          | 1          | true      | BTREE | A    | 2001
MY_TABLE        | I5                  | ID          | 2          | true      | BTREE | D    | 2001

14 rows selected
splice>
```

Show PrimaryKeys

The `show primarykeys` command displays all the primary keys in the specified table.

Syntax

```
SHOW PRIMARYKEYS FROM schemaName.tableName
```

schemaName

The schema name.

tableName

The table name.

Results

The `show primary keys` command results contains the following columns:

Column Name	Description
TABLE_NAME	The name of the table
COLUMN_NAME	The name of the column
KEY_SEQ	The order of the column within the primary key
PK_NAME	The unique name of the constraint

Examples

```
splice> create table myTable(i int, j int, primary key (i,j));
0 rows inserted/updated/deleted

splice> show primarykeys from mySchema.myTable;
TABLE_NAME | COLUMN_NAME | KEY_SEQ | PK_NAME
-----+-----+-----+-----
A       | I        | 1      | SQL141120202723310
A       | J        | 2      | SQL141120202723310
2 rows selected
```

Show Procedures

The `show procedures` command displays all of the procedures that have been created with the `create procedure` statement in the database or specified schema.

Syntax

```
SHOW PROCEDURES [ IN schemaName ]
```

schemaName

If you supply a schema name, only the procedures in that schema are displayed; otherwise, all procedures in the database are displayed.

Results

The `show procedures` command results contains the following columns:

Column Name	Description
PROCEDURE_SCHEMA	The name of the procedure's schema
PROCEDURE_NAME	The name of the procedure
REMARKS	Any remarks that have been stored for the procedure

Examples

PROCEDURE_SCHEM PROCEDURE_NAME		REMARKS	SYS
CS_UTIL	COLLECT_SCHEMA_STATISTICS	com.splicemachine. ...	
SYSCS_UTIL	COLLECT_TABLE_STATISTICS	com.splicemachine. ...	
SYSCS_UTIL	DISABLE_COLUMN_STATISTICS	com.splicemachine. ...	
SYSCS_UTIL	DROP_SCHEMA_STATISTICS	com.splicemachine. ...	
SYSCS_UTIL	DROP_TABLE_STATISTICS	com.splicemachine. ...	
SYSCS_UTIL	ENABLE_COLUMN_STATISTICS	com.splicemachine. ...	
SYSCS_UTIL	GET_ACTIVATION	com.splicemachine. ...	
SYSCS_UTIL	IMPORT_DATA	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_BACKUP_DATABASE	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_CANCEL_BACKUP	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_CANCEL_DAILY_BACKUP	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_COMMIT_CHILD_TRANSACTION	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_CREATE_USER	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_DELETE_BACKUP	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_DELETE_OLD_BACKUPS	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_DROP_USER	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_ELEVATE_TRANSACTION	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_EMPTY_STATEMENT_CACHE	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_ENABLE_ENTERPRISE	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_FLUSH_TABLE	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_ACTIVE_SERVERS	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_ACTIVE_TRANSACTION_IDS	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_ALL_PROPERTIES	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_AUTO_INCREMENT_ROW_LOCATIONS	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_CURRENT_TRANSACTION	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_GLOBAL_DATABASE_PROPERTY	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_LOGGERS	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_LOGGER_LEVEL	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_REGION_SERVER_CONFIG_INFO	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_REGION_SERVER_STATS_INFO	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_REQUESTS	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_SCHEMA_INFO	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_STORED_STATEMENT_PLAN_INFO	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_TIMESTAMP_GENERATOR_INFO	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_TIMESTAMP_REQUEST_INFO	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_VERSION_INFO	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_GET_WRITE_INTAKE_INFO	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_INVALIDATE_STORED_STATEMENTS	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_KILL_STALE_TRANSACTIONS	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_KILL_TRANSACTION	com.splicemachine. ...	
SYSCS_UTIL	SYSCS MODIFY_PASSWORD	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_RECOMPILE_INVALID_STORED_STATEMENTS	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_REFRESH_EXTERNAL_TABLE	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_RELOAD_SECURITY_POLICY	com.splicemachine. ...	
SYSCS_UTIL	SYSCS_RESET_PASSWORD	com.splicemachine. ...	

```
SYSCS_UTIL    | SYSCS_RESTORE_DATABASE          | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_RESTORE_DATABASE_OWNER   | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_SCHEDULE_DAILY_BACKUP    | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_SET_GLOBAL_DATABASE_PROPERTY | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_SET_LOGGER_LEVEL         | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_SET_USER_ACCESS          | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_SPLIT_REGION_AT_POINTS   | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_SPLIT_TABLE              | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_SPLIT_TABLE_AT_POINTS    | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_SPLIT_TABLE_EVENLY        | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_START_CHILD_TRANSACTION  | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_UPDATE_ALL_SYSTEM_PROCEDURES | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_UPDATE_METADATA_STORED_STATEMENTS | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_UPDATE_SCHEMA_OWNER       | com.splicemachine. ...
SYSCS_UTIL    | SYSCS_UPDATE_SYSTEM_PROCEDURE   | com.splicemachine. ...
SYSCS_UTIL    | UPSERT_DATA_FROM_FILE          | com.splicemachine. ...
SYSCS_UTIL    | VACUUM                          | com.splicemachine. ...
66 rows selected
```

Show Roles

The `show roles` command a sorted list of all of the roles defined in the database.

Syntax

```
SHOW ROLES
```

Examples

```
splice> create role testRole;  
0 rows inserted/updated/deleted  
splice> show roles;
```

ROLEID

TESTROLE

```
1 row selected
```

Show Schemas

The `show schemas` command displays all of the schemas in the current connection.

Syntax

```
SHOW SCHEMAS
```

Examples

```
splice> show schemas;
TABLE_SCHEM
-----
NULLID
SPLICE
SQLJ
SYS
SYSCAT
SYSCS_DIAG
SYSCS_UTIL
SYSFUN
SYSIBM
SYSPROC
SYSSTAT
11 rows selected
```

Show Synonyms

The `show synonyms` command displays all of the synonyms that have been created with the [CREATE SYNONYM](#) statement in the database or specified schema.

Syntax

```
SHOW SYNONYMS [ IN schemaName ]
```

schemaName

If you supply a schema name, only the synonyms in that schema are displayed; otherwise, all synonyms in the database are displayed.

Results

The `show synonyms` command results contains the following columns:

Column Name	Description
TABLE_SCHEMA	The name of the synonym's schema
TABLE_NAME	The name of the table
CONGLOM_ID	The conglomerate number, which points to the corresponding table in HBase
REMARKS	Any remarks associated with the table

Examples

```
splice> show synonyms;
TABLE_SCHEM | TABLE_NAME          | CONGLOM_ID | REMARKS
-----
SPICE       | HITTING           | NULL      |
1 rows selected
```

Show Tables

The `show tables` command displays all of the tables in the current or specified schema.

schemaName

If you supply a schema name, only the tables in that schema are displayed; otherwise, the tables in the current schema are displayed.

Syntax

```
SHOW TABLES [ IN schemaName ]
```

Results

The `show tables` command results contains the following columns:

Column Name	Description
TABLE_SCHEMA	The name of the table's schema
TABLE_NAME	The name of the table
CONGLOM_ID	The conglomerate number, which points to the corresponding table in HBase
REMARKS	Any remarks associated with the table

Examples

```
splice> show tables;
```

TABLE_SCHEM	TABLE_NAME	CONGLOM_ID	REMARKS
SYS	SYSALIASES	352	
SYS	SYSBACKUP	1040	
SYS	SYSBACKUPFILESET	1056	
SYS	SYSBACKUPITEMS	1136	
SYS	SYSBACKUPJOBS	1200	
SYS	SYSCHECKS	368	
SYS	SYSCOLPERMS	704	
SYS	SYSCOLUMNS	64	
SYS	SYSCOLUMNSTATS	1216	
SYS	SYSCONGLOMERATES	80	
SYS	SYSCONSTRAINTS	320	
SYS	SYSDEPENDS	240	
SYS	SYSFILES	256	
SYS	SYSFOREIGNKEYS	288	
SYS	SYSKEYS	272	
SYS	SYSPERMS	784	
SYS	SYSPHYSICALSTATS	1232	
SYS	SYSPRIMARYKEYS	384	
SYS	SYSROLES	736	
SYS	SYSROUTINEPERMS	720	
SYS	SYSSCHEMAPERMS	1392	
SYS	SYSSCHEMAS	32	
SYS	SYSSEQUENCES	752	
SYS	SYSSTATEMENTS	304	
SYS	SYSTABLEPERMS	688	
SYS	SYSTABLES	48	
SYS	SYSTABLESTATS	1248	
SYS	SYSTRIGGERS	656	
SYS	SYSUSERS	816	
SYS	SYSVIEWS	336	
SYSIBM	SYSDUMMY1	1344	
SPLICE	MYTABLE	1536	

32 rows selected

```
splice>show tables in SPLICE;
```

TABLE_SCHEM	TABLE_NAME	CONGLOM_ID	REMARKS
SPLICE	MYTABLE	1536	

1 row selected

Show Views

The `show views` command displays all of the views in the current or specified schema.

Syntax

```
SHOW VIEWS [ IN schemaName ]
```

schemaName

If you supply a schema name, only the views in that schema are displayed; otherwise, the views in the current schema are displayed.

Results

The `show views` command results contains the following columns:

Column Name	Description
TABLE_SCHEMA	The name of the table's schema
TABLE_NAME	The name of the table
CONGLOM_ID	The conglomerate number, which points to the corresponding table in HBase
REMARKS	Any remarks associated with the table

Examples

```
splice> show views;
TABLE_SCHEMA | TABLE_NAME | CONGLOM_ID | REMARKS
-----
SPICE | GUITAR_BRANDS | 4321 |
0 rows selected
```

Splice Machine Database Console Guide

This topic introduces the *Splice Machine Database Console*, a browser-based tool that you can use to monitor database queries on your cluster in real time. The Console UI allows you to see the Spark queries that are currently running in Splice Machine on your cluster, and to then drill down into each job to see the current progress of the queries, and to identify any potential bottlenecks. If you see something amiss, you can also terminate a query.



The *Splice Machine Database Console* leverages the Spark cluster manager *Web UI*, which is described here: <http://spark.apache.org/docs/latest/monitoring.html>.

This section is organized into the following topics:

- » The next section, [About the Splice Machine Database Console](#), tells you about the Database Console, including how to access it in your browser.
- » The Features of the Splice Machine Database Console topic describes how to use major features of the console interface.
- » The Managing Queries with the Console topic shows you how to review and monitor the progress of your Spark jobs.

About the Splice Machine Database Console

The *Splice Machine Spark Database Console* is a browser-based tool that you can use to watch your active Spark queries execute and to review the execution of completed queries. You can use the console to:

- » View any completed jobs
- » Monitor active jobs as they execute
- » View a timeline chart of the events in a job and its stages
- » View a Directed Acyclic Graph (DAG) visualization of a job's stages and the tasks within each stage
- » Monitor persisted and cached storage in realtime

How you access the Splice Machine Database Console depends on which Splice Machine product you're using:

Product	DB Console Access
Database-as-Service	<ul style="list-style-type: none"> » To monitor the Splice Machine jobs running on your cluster, click the DB Console button at the top right of your Management screen or click the DB Console link in the cluster created email that you received from Splice Machine. » To monitor any non-Splice Machine Spark jobs that are running on your cluster, you need to use a different Spark console, which you can access by clicking the External Spark Console link that is displayed in the bottom left corner of your cluster's dashboard page.

Product	DB Console Access
On-Premise Database	http://localhost:4040



The Database Console URL will only be active after you've run at least one query on our Spark engine; prior to using the Spark engine, your browser will report an error such as *Connection Refused*.

Here are some of the terms you'll encounter while using the Database Console:

Term	Description
<i>Accumulators</i>	Accumulators are variables programmers can declare in Spark applications that can be efficiently supported in parallel operations, and are typically used to implement counters and sums.
<i>Additional Metrics</i>	You can indicate that you want to display additional metrics for a stage or job by clicking the Show Additional Metrics arrow and then selecting which metrics you want shown.
<i>DAG Visualization</i>	A visual depiction of the execution Directed Acyclic Graph (DAG) for a job or job stage, which shows the details and flow of data. You can click the DAG Visualization arrow to switch to this view.
<i>Enable Zooming</i>	For <i>event timeline</i> views, you can enable zooming to expand the view detail for a portion of the timeline. You can click the Event Timeline arrow to switch to this view.
<i>Event Timeline</i>	A view that graphically displays the sequence of all <i>jobs</i> , a specific job, or a <i>stage</i> within a job.
<i>Executor</i>	A process that runs <i>tasks</i> on a cluster node.
<i>GC Time</i>	The amount of time spent performing garbage collection in a stage.
<i>Job</i>	The basic unit of execution in the Spark engine, consisting of a set of stages. With some exceptions, each query submitted to the Spark engine is a single job. Each job is assigned a unique Job Id and is part of a unique Job Group.
<i>Locality Level</i>	To minimize data transfers, Spark tries to execute as close to the data as possible. The <i>Locality Level</i> value indicates whether a task was able to run on the local node.

Term	Description
<i>Scheduling Mode</i>	<p>The scheduling mode used for a job.</p> <p>In FIFO scheduling, the first job gets priority on all available resources while its stages have tasks to launch. Then the second job gets priority, and so on.</p> <p>In FAIR scheduling, Spark assigns tasks between jobs in a round robin manner, meaning that all jobs get a roughly equal share of the available cluster resources. Which means that short jobs can gain fair access to resources immediately without having to wait for longer jobs to complete.</p>
<i>Scheduling Pool</i>	<p>The FAIR schedule groups jobs into pools, each of which can have a different priority weighting value, which allows you to submit jobs with higher or lower priorities.</p>
<i>ScrollInsensitive row</i>	<p>A row in a result set that is scrollable, and is not sensitive to changes committed by other transactions or by other statements in the same transaction.</p>
<i>Shuffling</i>	<p>Shuffling is the reallocation of data between multiple stages in a Spark job.</p> <p><i>Shuffle Write</i> is amount of data that is serialized and written at the end of a stage for transmission to the next stage. <i>Shuffle Read</i> is the amount of serialized data that is read at the beginning of a stage.</p>
<i>Stage</i>	<p>The Splice Machine Spark scheduler splits the execution of a <i>job</i> into stages, based on the RDD transformations required to complete the job.</p> <p>Each stage contains a group of tasks that perform a computation in parallel.</p>
<i>Task</i>	<p>A computational command sent from the application driver to an <i>executor</i> as part of a <i>stage</i>.</p>

See Also

- » User Interface Features of the Splice Machine Database Console
- » Managing Queries with the Console
- » [Using Spark Libraries with Splice Machine](#)

Features of the Splice Machine Database Console

This section summarizes the use of major features of the Database Console interface, including:

- » [Drilling Down](#)
- » [Switching Views](#)
- » [Hovering](#)
- » [Refreshing the View](#)
- » [Zooming the Timeline View](#)

Drilling Down

In general, you can click anything that displays in blue ([like this](#)) to drill down into a more detailed view. For example, clicking [Explain](#) in the following description from the completed jobs table will drill down into the job details for Job 113:

Job Id (Job Group)	Description
113 (SPLICE <387>)	-- EXPLAIN QUERY 22 explain select cntrycode, count(*) as numcust, sum(c_acctbal) as tota... Explain

You can continue to drill down from there to reveal increasing levels of detail.

Switching Views

You can quickly switch to a different view by clicking a tab in the tab bar at the top of the console screen. The Jobs tab is selected in this screen shot:



Hovering

You can hover the cursor over interface element links, like the [Event Timeline](#) drop-down in the following image, to display a screen tip for the item:

Spark Jobs (?)

Total Uptime: 53 min

Scheduling Mode: FAIR

Completed Jobs:

Shows when jobs started and ended and when executors joined or left. Drag to scroll. Click Enable Zooming and use mouse wheel to zoom in/out.

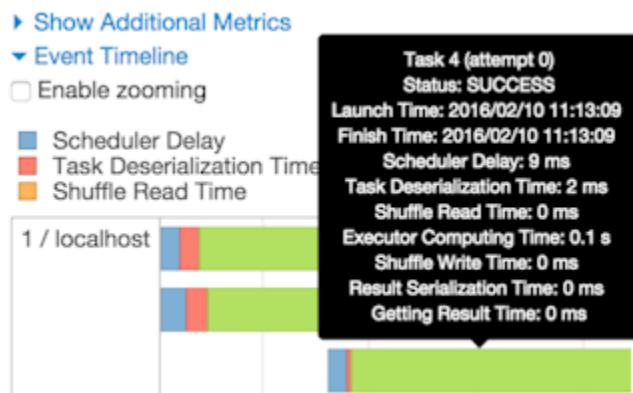
Completed Jobs

Similarly, you can hover over the ? to display the definition for a term, like the definition of a job:

Spark Jobs (?)

A job is triggered by an action, like count() or saveAsTextFile(). Click on a job to see information about the stages of tasks inside it.

And you can hover over an event in timeline display to see summary information; for example:



Refreshing the View

Currently, the console does not automatically or periodically refresh the view.

If you're monitoring an active job, you'll need to refresh your browser window to view the latest activity.

Zooming the Timeline View

When you're viewing an event timeline, you can **Enable zooming**, which allows you to use mouse or touch gestures to zoom in on a portion or a timeline, zoom out, or scroll through the timeline.

See Also

- » About the DB Console
- » Managing Queries with the DB Console

Managing Queries with the DB Console

The Splice Machine Database Console allows you to view queries that are currently running and have completed running in your database. You typically start at the top level, viewing jobs, and then drill down into individual job details, job stages, and task details, as described in these sections:

- » [Viewing Summary Pages](#) describes the console's top-level summary pages.
- » [Viewing Job Details](#) describes the pages in which you can view details of active or completed jobs.
- » [Viewing Stage Details](#) describes the pages in which you can view details of active and completed stages.
- » [Terminating a Stage](#) shows you how to terminate a job stage that is not performing as you think it should.

Viewing Summary Pages

The console includes five summary pages, each of which can be accessed from the tab bar at the top of the console window:

- » [The Jobs Summary page](#) shows information about all active and completed jobs.
- » [The Stages Summary Page](#) shows all stages for all jobs, both active and completed.
- » [The Storage Summary Page](#) shows any RDDs that you have persisted or cached to memory.
- » [The Environment Summary Page](#) shows information about the Spark run-time environment.
- » [The Executors Summary Page](#) shows the executors that are currently running.

The Jobs Summary page

The *Jobs Summary Page* is the top-level view in the Splice Machine Database Console. It shows you a summary of any currently active and all completed jobs.

You land on this page when you first view the *Database Console* in your browser, and you can view it at any time by clicking the *Jobs* tab in the tab bar at the top of the page.

Spark Jobs [\(?\)](#)

Total Uptime: 26 min
Scheduling Mode: FAIR
Active Jobs: 1
Completed Jobs: 49

[▶ Event Timeline](#)

Active Jobs (1)

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
49 (SPLICE <272>)	Sort: Shuffle/Sort Data	2016/02/10 15:07:58	6 s	0/4	 0/16

Completed Jobs (49)

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
48 (SPLICE <268>)	-- QUERY 04 select o_orderpriority, count(*) as order_count from tpch.orders where o_orderd... Produce Result Set	2016/02/10 14:58:32	38 ms	1/1 (5 skipped)	 1/1 (21 skip)
47 (SPLICE <268>)	-- QUERY 04 select o_orderpriority, count(*) as order_count from tpch.orders where o_orderd... Produce Result Set	2016/02/10 14:58:32	41 ms	1/1 (5 skipped)	 1/1 (21 skip)
46 (SPLICE <268>)	-- QUERY 04 select o_orderpriority, count(*) as order_count from tpch.orders where o_orderd... Produce Result Set	2016/02/10 14:58:31	39 ms	1/1 (5 skipped)	 1/1 (21 skip)
45 (SPLICE <268>)	-- QUERY 04 select o_orderpriority, count(*) as order_count from tpch.orders where o_orderd... Produce Result Set	2016/02/10 14:58:31	39 ms	1/1 (5 skipped)	 1/1 (21 skip)
44 (SPLICE <268>)	-- QUERY 04 select o_orderpriority, count(*) as order_count from tpch.orders where o_orderd... Produce Result Set	2016/02/10 14:58:31	0.1 s	2/2 (4 skipped)	 6/6 (16 skip)
43 (SPLICE <268>)	-- QUERY 04 select o_orderpriority, count(*) as order_count from tpch.orders where o_orderd... Sort: Shuffle/Sort Data	2016/02/10 14:58:01	31 s	5/5	 21/21
42 (SPLICE <267>)	-- QUERY 03 select l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue, o_orderdat... Produce Result Set	2016/02/10 14:57:06	54 s	5/5	 5/5
41 (SPLICE)	-- QUERY 01 select l_returnflag, l_linenumber, sum(l_quantity) as sum_qty, sum(l_extendedpric... ...	2016/02/10	37 ms	1/1 (2 skipped)	 1/1 (10 skip)

NOTE: A stage is shown as skipped when the data has been fetched from a cache and there was no need to reexecute the stage; this happens when shuffling data because the Spark engine automatically caches generated data.

You can click the a job description name (in blue) to view job details of any job in the Active Jobs or Completed Jobs sections.

The Stages Summary Page

The *StagesSummary Page* shows you the available scheduling pools, and a summary of the stages for all active and completed jobs. You can access this page by clicking the **Stages** tab in the tab bar at the top of the window.

The screenshot shows the SpliceMachine application UI with the following details:

- Header:** Shows the Spark 1.5.0 logo, Jobs, Stages, Storage, Environment, Executors, and the IP address 192.168.1.15.
- Section: Stages for All Jobs**
 - Active Stages:** 2
 - Pending Stages:** 3
 - Completed Stages:** 93
- Section: 6 Fair Scheduler Pools**

Pool Name	Minimum Share	Pool Weight	Active Stages	Running Tasks	SchedulingMode
import	0	10	0	0	FAIR
query	0	20	2	4	FAIR
admin	0	1	0	0	FAIR
compaction	0	5	0	0	FAIR
urgent	0	1000	0	0	FAIR
default	0	1	0	0	FIFO
- Section: Active Stages (2)**

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
183	query	-- QUERY 08 select o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(... Merge Sort Join: Prepare Left Side (kill) 15:10:26	2016/02/10	8 s	0/5				
182	query	-- QUERY 08 select o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(... Merge Sort Join: Prepare Right Side (kill) 15:10:26	2016/02/10	8 s	0/1				
- Section: Pending Stages (3)**

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
186		Grouped Aggregate: Prepare Keys	Unknown	Unknown	0/5				
187		Sort: Shuffle/Sort Data	Unknown	Unknown	0/5				
185		Merge Sort Join: Prepare Left Side	Unknown	Unknown	0/5				

You can click the descriptive name of a stage (in blue) to view the stage details.

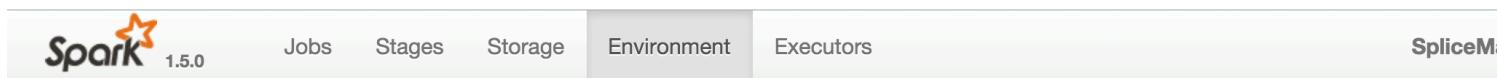
The Fair Scheduler Pools section at the top of the page shows the name and weighting value for each of the scheduler pools that have been defined for your database jobs.

The Storage Summary Page

The *Storage Summary Page* displays information about any RDDs that are currently persisted or cached. You can access this page by clicking the Storage tab in the tab bar at the top of the window:

The Environment Summary Page

The *Environment Summary Page* displays information about which software versions you're using, and shows the values of the Spark-related environment variables. You can access this page by clicking the Environment tab in the tab bar at the top of the window:



Environment

Runtime Information

Name	Value
Java Home	/Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Home/jre
Java Version	1.7.0_71 (Oracle Corporation)
Scala Version	version 2.10.4

Spark Properties

Name	Value
driver.source.splice-machine.class	com.splicemachine.derby.stream.spark.SpliceMachineSource
executor.source.splice-machine.class	com.splicemachine.derby.stream.spark.SpliceMachineSource
spark.app.id	application_1454971247434_0001
spark.app.name	SpliceMachine
spark.driver.appUIAddress	http://192.168.1.15:4040
spark.driver.cores	1
spark.driver.extraClassPath	/Users/garyhillerson/git/spliceengine/cdh5.4.1/splice_machine_te llerson/git/spliceengine/cdh5.4.1/splice_machine_test/target/dep
spark.driver.host	localhost
spark.driver.maxResultSize	1g
spark.driver.memory	1g
spark.driver.port	58937
spark.enabled	true
spark.executor.cores	4

The Executors Summary Page

The *Executors Summary Page* shows you the Spark executors that are currently running. You can access this page by clicking the **Executors** tab in the tab bar at the top of the window:

Executors (2)

Memory: 0.0 B Used (912.8 MB Total)

Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
1	localhost:50141	0	0.0 B / 176.7 MB	0.0 B	0	8	30	38	11.0 s	709.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
driver	192.168.1.15:50132	0	0.0 B / 736.1 MB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B		Thread Dump

You can click **Thread Dump** to display a thread dump for an executor, or you can click a log name to see the contents of the log.

Viewing Job Details

If you click a job to see its details, you'll see a screen like the following displayed, which shows the stages of the job:



Details for Job 157

Status: SUCCEEDED

Job Group: SPLICE <1431>

Completed Stages: 7

▶ [Event Timeline](#)

▶ [DAG Visualization](#)

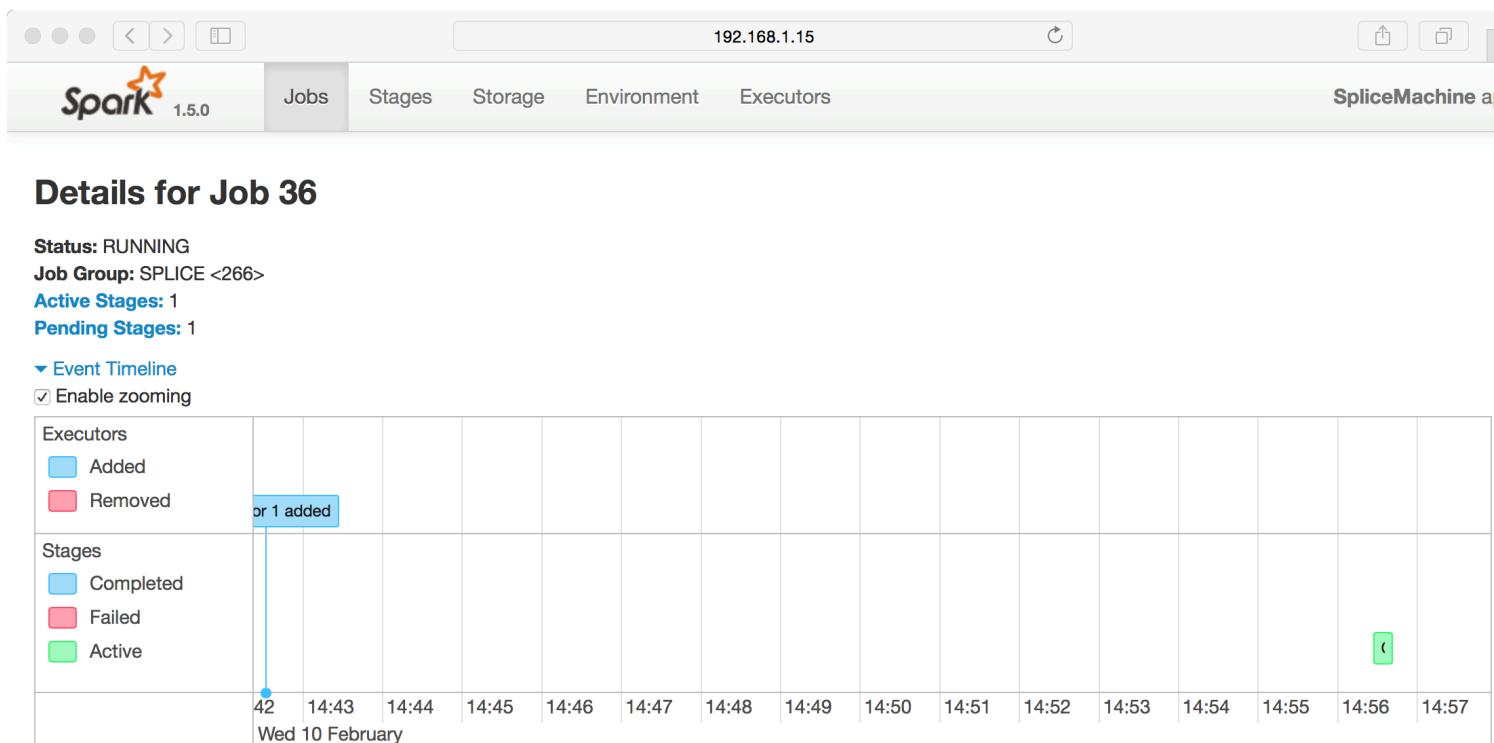
Completed Stages (7)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shu
163	query	-- QUERY 02 select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_c... Produce Result Set	2016/02/08 15:37:23	32 ms	1/1			18.9 KB
162	query	-- QUERY 02 select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_c... Row Limit: Fetch Limit 100	2016/02/08 15:37:23	45 ms	1/1			
161	query	-- QUERY 02 select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_c... Row Limit: Fetch Limit 100	2016/02/08 15:37:23	0.1 s	1/1			105.9 KB
160	query	-- QUERY 02 select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_c... Sort: Prepare Keys	2016/02/08 15:37:22	1 s	1/1			6.3 MB
158	query	-- QUERY 02 select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_c... Merge Sort Join: Prepare Right Side	2016/02/08 15:37:20	1 s	1/1			6.2 MB
159	query	-- QUERY 02 select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_c... Merge Sort Join: Prepare Left Side	2016/02/08 15:37:05	13 s	1/1	20.3 MB		
157	query	-- QUERY 02 select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_c... Grouped Aggregate: Prepare Keys	2016/02/08 15:37:05	16 s	1/1	52.3 MB		

You can expand the job detail display by selecting the [Event Timeline](#) and/or [DAG Visualization](#) buttons.

Job Details Event Time Line View

The job details time-line view looks like the following screen shot:



► DAG Visualization

Active Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output
53	query	-- QUERY 01 select l_returnflag, l_linenumber, sum(l_quantity) as sum_qty, sum(l_extendedprice... Grouped Aggregate: Prepare Keys	2016/02/10 14:56:26	15 s	0/5	325.4 MB	

Pending Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
54		Sort: Shuffle/Sort Data	Unknown	Unknown	0/5				

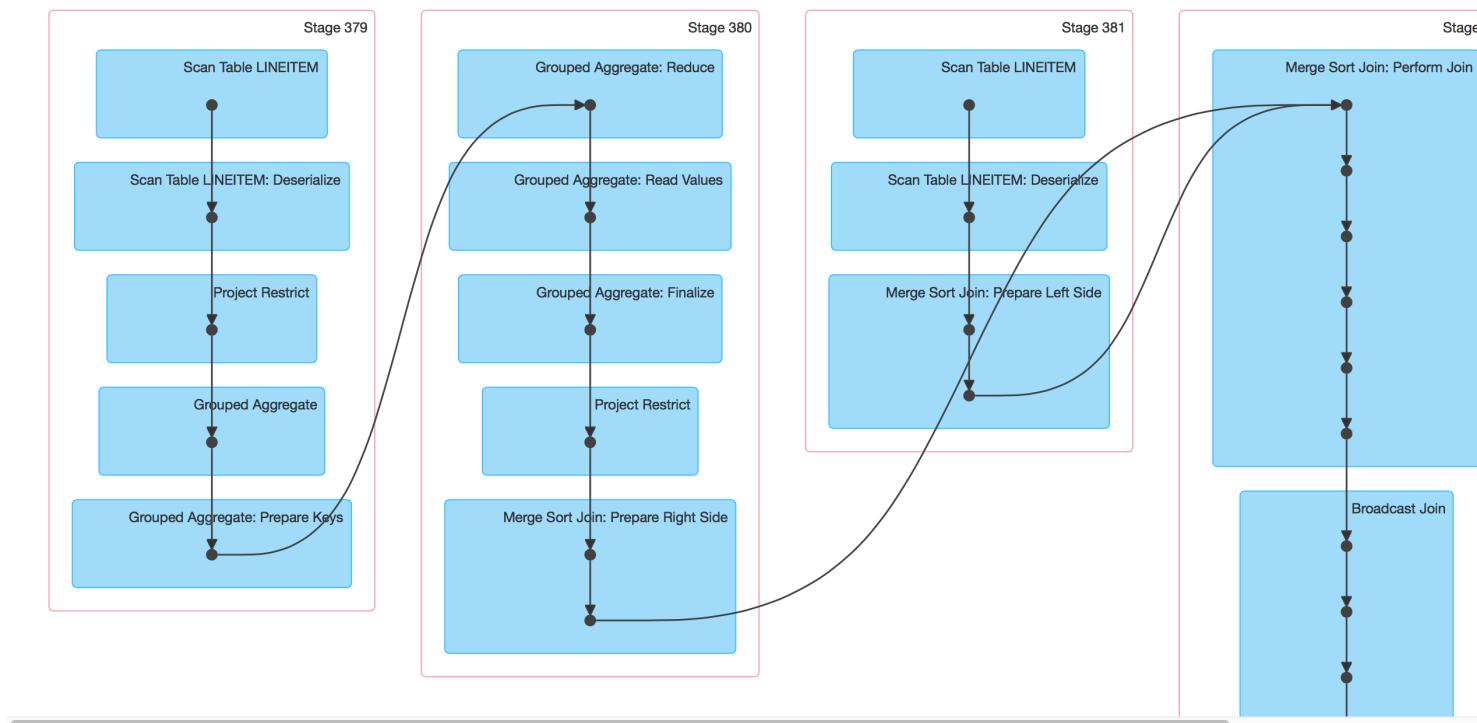
Job Details Graphical Visualization View

The DAG Visualization view for a job looks like this:

Details for Job 188

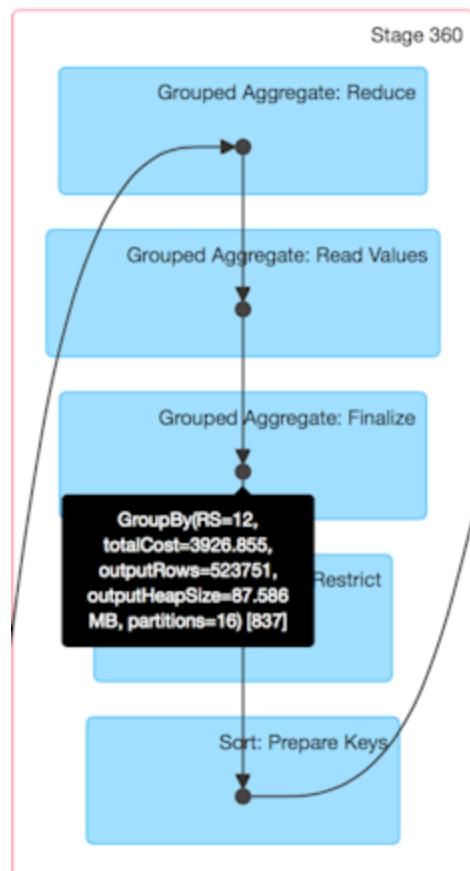
Status: RUNNING
Job Group: SPICE <684>
Active Stages: 1
Pending Stages: 1
Completed Stages: 3

- ▶ Event Timeline
- ▼ DAG Visualization



Some key things to know about the DAG view are:

- » You can click in the box representing a stage to view the detailed tasks within that stage. For an example, see [Graphical View of the Tasks in a Stage](#), in the next section.
- » You can hover over any of the black dots inside a task box to display information about the task. For example:



Viewing Stage Details

Viewing stage details is very much the same as viewing job details. If you click the name of a stage in another page, the detailed view of that stage displays:

 1.5.0	Jobs	Stages	Storage	Environment	Executors	SpliceMachine
---	------	--------	---------	-------------	-----------	---------------

Details for Stage 134 (Attempt 0)

Total Time Across All Tasks: 2 s

Input Size / Records: 10.0 KB / 0

- ▶ [DAG Visualization](#)
- ▶ [Show Additional Metrics](#)
- ▶ [Event Timeline](#)

Summary Metrics for 16 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	61 ms	0.1 s	0.1 s	0.2 s	0.3 s
Scheduler Delay	8 ms	9 ms	9 ms	10 ms	12 ms
Task Deserialization Time	1 ms	2 ms	2 ms	9 ms	10 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Result Serialization Time	0 ms	1 ms	1 ms	1 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Input Size / Records	0.0 B / 0	0.0 B / 0	0.0 B / 0	0.0 B / 0	3.9 KB / 0

Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Input Size / Records
1	localhost:51853	3 s	16	0	16	10.0 KB / 0

Tasks

Index						Executor	Launch		Scheduler	Task Deserialization	GC	Result Serialization	Getting Result	Peak Execution

The Event Time Line View of a Stage

The Event Timeline view of a stage looks like this:

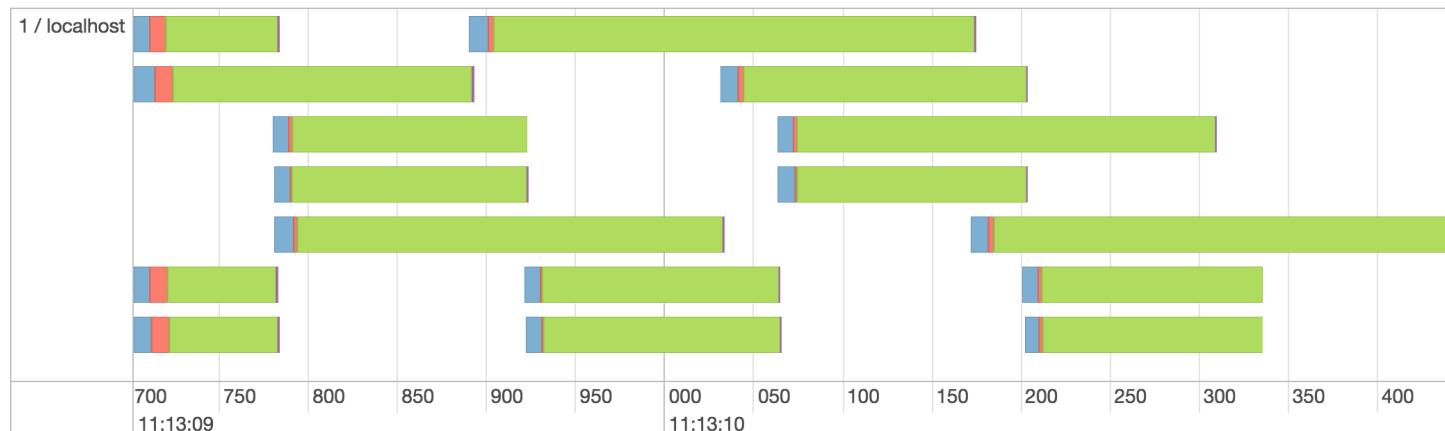
[▶ Show Additional Metrics](#)

[▼ Event Timeline](#)

Enable zooming

Scheduler Delay
Task Deserialization Time
Shuffle Read Time

Executor Computing Time
Shuffle Write Time
Result Serialization Time

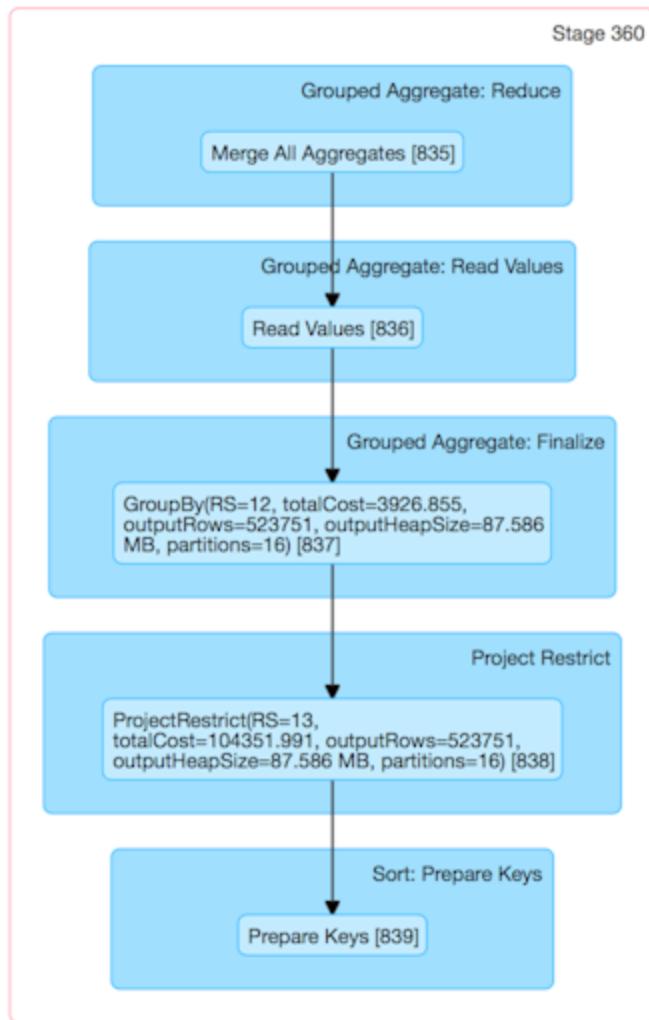


Summary Metrics for 16 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	61 ms	0.1 s	0.1 s	0.2 s	0.3 s
Scheduler Delay	8 ms	9 ms	9 ms	10 ms	12 ms
Task Deserialization Time	1 ms	2 ms	2 ms	9 ms	10 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Result Serialization Time	0 ms	1 ms	1 ms	1 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Input Size / Records	0.0 B / 0	0.0 B / 0	0.0 B / 0	0.0 B / 0	3.9 KB / 0

Graphical View of the Tasks in a Stage

The DAG Visualization view of a stage looks like this:



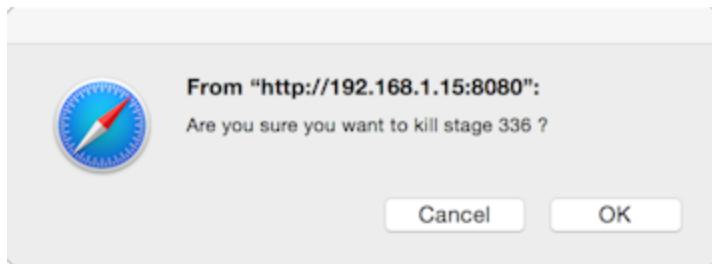
Terminating a Stage

If you conclude that an active job stage is not performing the way you think it should, you can terminate a stage by clicking the Kill button shown in the description of every active stage. The following image highlights the kill buttons that you'll find in the console display:

Active Stages (2)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
183	query	-- QUERY 08 select o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(...) Merge Sort Join: Prepare Left Side	2016/02/10	8 s	0/5				
182	query	-- QUERY 08 select o_year, sum(case when nation = 'BRAZIL' then volume else 0 end) / sum(...) Merge Sort Join: Prepare Right Side	2016/02/10	8 s	0/1				

You'll be prompted to verify that you want the stage terminated:



You can access the Kill button by drilling down into a job's stages, or by selecting the Stages tab in the tab bar, which displays all stages for all jobs.

See Also

- » [About the Splice Machine Database Console](#)
- » [User Interface Features of the Splice Machine Database Console](#)
- » [Using Spark Libraries with Splice Machine](#)

Welcome to the Splice Machine Database Service!

Welcome to Splice Machine, the database platform for adaptive applications that manage operational processes. This site contains documentation for our [Managed Database Service in the Cloud](#), which includes Release 2.7 of the Splice Machine Database.

Getting Started With our Database Service

Getting started with our database is as simple as can be; just follow these steps, and you can be up and running in less than an hour:

- 1 LOG INTO SPICE MACHINE** → Log in directly, or use your Google or Amazon ID.
- 2 CREATE DATABASE CLUSTER** → Adjust 4 sliders for your processing and storage needs; your database cluster is ready within 15 minutes.
- 3 LOAD YOUR DATA** → Copy data to S3, then perform a fast import. Time required varies with dataset size. Our [Zeppelin Simple Example](#) provides a quick example.
- 4 QUERY AND UPDATE YOUR DATABASE** → Use Zeppelin notebooks to quickly update, query, and display results graphically, without coding.

Next Steps

Easy next steps you can take to become more proficient with your new database system:

- » Our About the Splice Machine Database Service topic introduces this edition of Splice Machine and links to main documentation pages related to the service.
- » Spend some time learning more about creating and using Zeppelin notebooks, which you can use to prepare and run SQL DDL and DML, stored procedures, Java, Scala, and Python and Spark-SQL programs with Splice Machine data, all without writing code.
- » Spend a few minutes with our Cloud Manager Interface, which you can use to modify your cluster configuration, administer your account, set up events, and review database usage.
- » Check this documentation web for best practices, usage tips, developer guides, and reference material

About the Splice Machine Database Service

With the *Splice Machine Cloud Manager*, configuring a new cluster is as easy as using a few sliders to set compute units for OLTP and OLAP processing, allocate storage, and schedule backup frequency and retention. Splice Machine does the rest. You can seamlessly scale out from gigabytes to petabytes of data when needs or data volumes change, and the same configurator adds or subtracts resources dynamically. You pay only for what you use. You can then:

- » Power your applications on a scale-out, ANSI SQL database
- » Power apps with simultaneous OLAP & OLTP workloads
- » Ingest millions of records and process thousands of transactions in nanoseconds
- » Elastically scale resources as needed
- » We've got you covered – availability, backups, monitoring and alerts

Service Configuration

You only need to understand a few key concepts to configure your service:

- » A **Splice Unit** is a measure of processing work; one unit currently translates (approximately) to 2 virtual CPUs and 16 GB of memory. When you provision a new Splice Machine cluster, you can select the number of Splice Units you want to use for OLAP and OLTP workloads. The minimum number of Splice Units required for your cluster changes when you update the amount of data you want to access or the amount processing power you want to use.
- » The space allocated for your **Internal Dataset**, which is data that you're storing within your database. Note that as this size increases, the number of Splice Units required (especially OLTP Splice Units) can also increase.
- » The space allocated for your **External Dataset**, which is data stored externally that you can access from your database using features such as external tables and VTI. Note that as this size increases, the number of OLAP Splice Units required can also increase.

Use our Database-Service documentation to quickly walk through provisioning your database cluster, loading your data, and querying your data in notebooks, all in less than an hour. Once you're ready, our documentation offers:

Available Tools

In addition to easy connectivity with almost any Business Intelligence tool, Splice Machine includes:

- » An integrated *Zeppelin Notebook* interface. Zeppelin notebooks are like text documents, but with code that can be executed and of which the output can be rendered as tables, reports and beautiful graphs. This enables you to prepare and run SQL DDL and DML, stored procedures, Java, Scala, and Python and Spark-SQL programs with Splice Machine data. Splice Machine comes pre-configured with a set of notebooks to get started, load data and see examples of the work that can be done with the RDBMS.
- » Our JDBC and ODBC drivers allow you to connect third-party business intelligence tools to your database.
- » You can also take advantage of machine learning, streaming, and other services that you can access from our predesigned notebooks, your own notebooks, or code written by your developers.

Learn More

Our documentation provides:

- » Complete descriptions of our Cloud Manager dashboard
- » Numerous Tutorials about connecting with other tools, using various programming languages with Splice Machine, ingesting data efficiently, and so on.
- » An introduction to Using Zeppelin with Splice Machine
- » A wealth of [Developer's Guide information](#) and our SQL Reference Manual

You can visit our company web site to learn about what our [Cloud-Managed Database-as-Service \(DBaaS\)](#) can do for your company.

Service Overview

Our Database Service is a subscription-based service, hosted in the Cloud. We take care of managing your cluster services, and you can focus on working with our scalable, dual-engine database.

This is a DB-Service-Only topic! [Learn about our products](#)

Service Availability

Splice Machine's target Service availability commitment is 99.9% per calendar month, excluding scheduled downtime. You can expect the following:

- » Splice Machine will deliver product updates with minimal, scheduled downtime.
- » Splice Machine can recover your database from a stored backup after receiving your request to do so.
- » Splice Machine can resize your cluster with minimal downtime.

Support for Your Service

Splice Machine provides two support options, as shown in the following table:

Support Type	Pricing	Support Feature	Description	Details
Standard Support	Free	Coverage Hours	Monday-Friday 9am-6pm Pacific time	(subject to local holidays)
		System Impaired	Significant issues with speed, quality, or functionality of Service.	< 12 business hours
		Other Issues	General queries and guidance requests	< 24 business hours
Business Support	As per contract Includes SLA	Coverage Hours	24 hours a day, 7 days a week, 365 days a year	
		Production System Down	Complete loss of Service on Production cluster.	< 1 hour
		Production System Impaired	Significant issues with speed, quality, or functionality of Service on Production cluster.	< 4 hours
		Production System Down	Significant issues with speed, quality, or functionality of Service on non-production cluster.	< 12 hours

Support Type	Pricing	Support Feature	Description	Details
		Other Issues	General queries and guidance requests	< 24 business hours

Service Level Agreement (SLA)

Our *business support agreement* includes a *Service Level Agreement (SLA)* that specifies our commitment to a target Service availability of 99.9% per calendar year, excluding scheduled downtimes.

Service Terms

Subscription fees are payable monthly in advance on the 1st of the month, pro-rated for any partial months. We'll charge your credit card or withdraw payment by ACH on the first of each month, until your service is cancelled. See your license agreement for more details.

Database Service User Interface

This is a DB-Service-Only topic! [Learn about our products](#)

In addition to our database, the Splice Machine Database Service includes all of the tools you need to create your cluster, load data into your database, query and manipulate your database, and create visual representations of your query results, as described here:

UI Component	Description
Dashboard	The Splice Machine Dashboard or Cloud Manager is your entry point to your Database Service. You can register and log into your account here, as well as accessing the other managers described in this table.
Cluster Manager	Use the Cluster Manager to create new clusters and to monitor the health of your clusters.
Notebooks Manager	Apache Zeppelin notebooks make it easy to query your database and apply various visualizations to the results of your queries. We've created several notebooks that will help you to quickly become productive and to see how easy it is to create your own notebooks.
Database Console	The Database Console is a browser-based tool that you can use to monitor database queries on your cluster in real time. The Console UI allows you to see the Spark queries that are currently running in Splice Machine on your cluster, and to then drill down into each job to see the current progress of the queries, and to identify any potential bottlenecks. If you see something amiss, you can also terminate a query.
Events Manager	Our Events Manager allows you to examine events that have occurred on your cluster.
Account Manager	The Splice Machine Account Manager is where you manage your users, your profile, and your billing information.

Release Notes for the Splice Machine Database-as-a-Service Product

This is a DB-Service-Only topic! [Learn about our products](#)

This topic includes any release notes that are specific to the Splice Machine *Database-as-Service* product, in these sections:

- » [Features Not Yet Available](#)
- » [Current Limitations](#)
- » [Important Notes](#)

Most of the information about changes in the Splice Machine database that underlies this product are found in the Splice Machine database release notes.

Features Not Yet Available

These features are not yet available, but will be very soon:

- » VPC Settings are not yet enabled but will be in a near future release.
- » You currently cannot cancel queries that are running through Zeppelin or JDBC tools; you can use the Spark User Interface to cancel Spark queries.

Current Limitations

These limitations exist in this release, and will be removed in the near future:

- » On a JDBC connection, individual queries or actions will time out after one hour; you can run long-running queries within a Zeppelin notebook.
- » Updating of CPU, Memory, and Disk usage graphs for clusters is currently limited: the updates are happening only intermittently.

Important Notes

These are important notes about issues you need to be aware of when using our Database Service:

- » The timestamps displayed in Zeppelin will be different than the timestamps you see in the Splice Machine Spark User Interface, depending upon your time zone.
- » Although Splice Machine backs up your database regularly, it does not back up your Zeppelin Notebook changes; please export your Notebooks regularly if you make changes.

Splice Machine Cloud Manager

This guide helps you to get registered with and start using the Splice Machine Cloud Manager



You can initiate a chat session with one of our support specialists by clicking this chat icon, which you'll see on Cloud Manager screens.

Here are the topics included in this guide:

Topic	Description
<i>Navigating Your Dashboard</i>	Your Dashboard is the entry point to the Splice Machine Cloud Manager. From here, you can create new clusters, access existing clusters, manage your account, review notifications, update your profile, and log out.
<i>Registering</i>	Shows you how to complete the first step of using your database service: registering as a user of the Splice Machine Cloud Manager.
<i>Logging In</i>	Shows you how to log into the Splice Machine Cloud Manager once you've registered.
<i>Creating a New Cluster</i>	Follow this steps in this topic to quickly become productive with your clustered database. In only a few minutes, you'll have your cluster up and running, and will be able to load and work with your data.
<i>Loading Your Data</i>	<p>You can load your data into Splice Machine from an AWS S3 bucket:</p> <ul style="list-style-type: none"> » If you don't yet know how to create an S3 bucket or upload data to a bucket, please check our Uploading Data to an S3 Bucket tutorial. » You may need to configure IAM permissions to allow Splice Machine to access your bucket; see our Configuring an S3 Bucket for Splice Machine Access tutorial. » Once you've got your data in a bucket, you can follow our Importing Data Tutorial to load that data into Splice Machine. <p>Also note that the S3 directory you specify for the log of record import issues (the <i>bad record</i> directory), must be write-accessible using the same AWS credentials that apply to the input directory, i.e. the bad record directory must allow the same access key/secret key pair.</p>

Topic	Description
<i>Using the DB Console</i>	<p>The Splice Machine Database Console is a browser-based tool that you can use to monitor database queries on your cluster in real time. The Console UI allows you to see the Spark queries that are currently running in Splice Machine on your cluster, and to then drill down into each job to see the current progress of the queries, and to identify any potential bottlenecks. If you see something amiss, you can also terminate a query.</p> <p>The DB Console is available for all Splice Machine products; you access the Splice DB Console for the Database-as-Service product by clicking the DB Console link in your Cluster Management dashboard, or by following the link sent to you by Splice Machine when your cluster was originally created.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  The DB Console link in your Cluster Management dashboard allows you to monitor Splice Machine jobs running on your cluster. If you are running non-Splice jobs, you'll need to use a different console to monitor them; you'll find a link to this External Spark Console in the bottom left corner of your cluster dashboard. </div>
<i>Using the Notebooks Manager</i>	<p>One of the great features of our database service is the ease with which you can use Apache Zeppelin notebooks to interact with your database. Our Notebooks Manager provides convenient access to your notebooks, and to information about using Zeppelin.</p> <p>This tutorial walks you through using a notebook created by Splice Machine that:</p> <ul style="list-style-type: none"> » Creates a schema and the tables in your database to store the TPCH-1 benchmarks data. » Loads the data from an S3 bucket into your database. » Runs any or all of the TPCH-1 queries
<i>Managing Your Account</i>	Use our Account Manager to manage your company profile, billing, and users.
<i>Reviewing Event Notifications.</i>	Use our Events Manager to review notification messages that have been sent to your account.

Cloud Manager User Registration

This page describes the Splice Machine Cloud Manager registration page.

The registration screen is straightforward and familiar:



Set up Splice Machine in minutes and start powering your applications on a scale-out, ANSI SQL database.

- ⊕ Full ANSI SQL database
- ⊕ Power apps with simultaneous OLAP & OLTP workloads
- ⊕ Ingest millions of records and process thousands of transactions in seconds
- ⊕ Scale up or scale down when you need it
- ⊕ We've got you covered - availability, backups, monitoring and alerts

Let's get started...

First Name	<input type="text" value="Enter First Name"/>	
Last Name	<input type="text" value="Enter Last Name"/>	
Email	<input type="text" value="Enter Email"/>	
Confirm Email	<input type="text" value="Re-enter Email"/>	
Password	<input type="text" value="Enter Password"/>	
Confirm Password	<input type="text" value="Re-enter Password"/>	
<input style="background-color: orange; color: white; padding: 5px 10px; margin-right: 10px" type="button" value="Register"/>		
Or register with...		
G Google		
A Amazon		

You can use your Google or Amazon account information to register, or you can manually register by following these steps:

1. Enter your first and last name.
2. Enter and then confirm the email address you want to use for logging into the Splice Machine Cloud Manager.
3. Enter and then confirm the password you want to use for logging into the Splice Machine Cloud Manager.
4. Click the **Register** button.

Once you've successfully registered, you'll land on the *Create New Cluster* screen; we've created a tutorial to quickly walk you through Creating a Cluster topic.

Logging in to the Splice Machine Cloud Manager

This page describes the Splice Machine Cloud Manager Login page.



Set up Splice Machine in minutes and start powering your applications on a scale-out, ANSI SQL database.

- ⊕ Full ANSI SQL database
- ⊕ Power apps with simultaneous OLAP & OLTP workloads
- ⊕ Ingest millions of records and process thousands of transactions in seconds
- ⊕ Scale up or scale down when you need it
- ⊕ We've got you covered - availability, backups, monitoring and alerts

Login

Email

Password

Register
Login

[Login with Google](#)
 [Login with Amazon](#)

[Forgot password?](#)

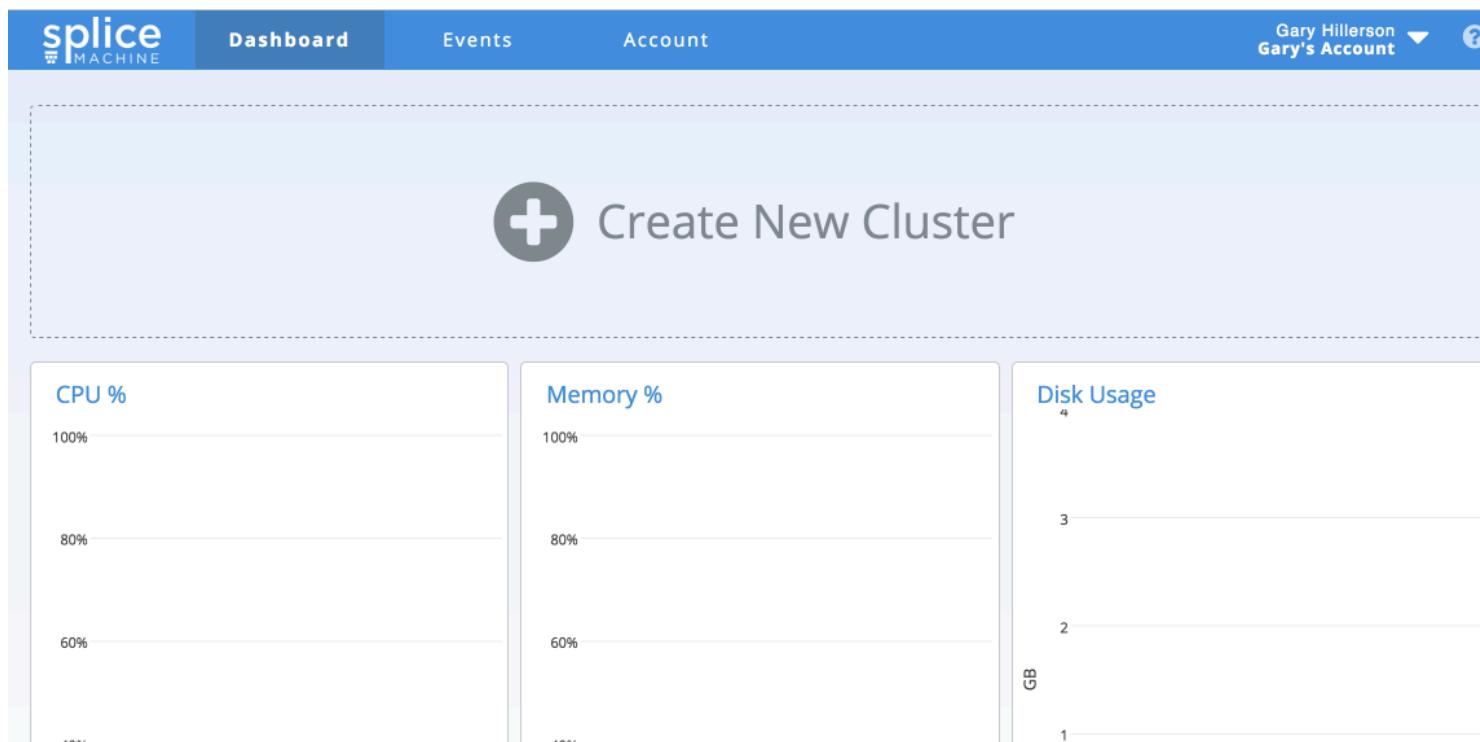
If you've already registered with the Splice Machine Cloud Manager, you can enter your registered email address and password, and click the **Login** button; alternatively, you can log in using your Google or Amazon credentials.

If you've not yet registered with the Splice Machine Cloud Manager, you'll need to register, which will also log you in. Click the **Register** button and fill in the registration form.

After successfully logging in, you'll land on your dashboard page.

Creating a New Splice Machine Cluster

When you first visit your new Splice Machine Cloud Manager dashboard, you'll see the initial dashboard view, which prompts you to create a new cluster:



Click **Create New Cluster** to start the process of provisioning your Splice Machine cluster. You'll then need to:

1. [Configure Cluster Parameters](#) for data sizing, cluster power, and backup frequency.
2. [Configure Cluster Access](#) for your users.
3. [Set Up Payment](#) for your Splice Machine cluster.
4. Start Using Splice Machine!

Configure Cluster Parameters

You use the **Create New Cluster** screen to provision your cluster:

splice
MACHINE

Dashboard
Events
Account
Account ▾

Create New Cluster

Region: us-east-1

Data Sizing

Internal Dataset (TB)

0.1 TB
5
10
25
50
100 TB

0.1
▲
▼

External Dataset (TB)

0 TB
50
100
250
500
1 PB

0
▲
▼

Cluster Power

OLTP Splice Units

4
25
50
100
250
500
600

4
▲
▼

OLAP Splice Units

4
25
50
100
250
500
600

4
▲
▼

Backup Frequency

Frequency:

Start Window (UTC):

Backups to Keep:

Estimated Benchmarks

Single Record Lookup (ms)
20ms

Import Rate (record/sec)
40,000

TPC-C 100 Users (TpmC)
4,000

TPC-H Query #2 (sec)
25

Estimated Costs (Monthly)

* Does not include monthly I/O costs.

●	Storage	\$85.00
●	Compute	\$3,558.00

Total
\$3,643.00

Reset
Back
Next >

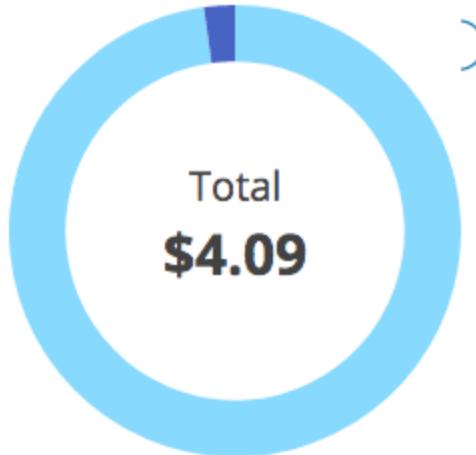
If you have subscribed to Splice Machine via the AWS Marketplace, your costs will be estimated on an hourly basis instead of a monthly basis:

Creating a New Splice Machine Cluster

1229

Estimated Costs (Hourly)

 Storage	\$0.09
 Compute	\$4.00



* Does not include monthly I/O costs.

Screen Help

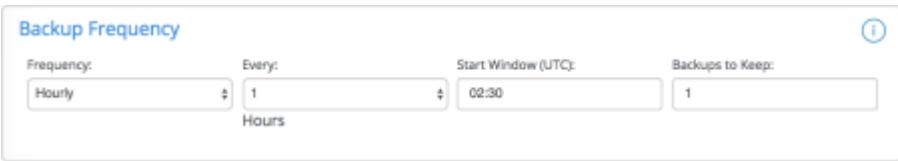
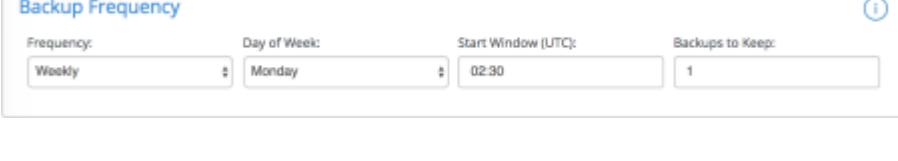
Many of the components of the [Create Cluster](#) screen, like most of our Cloud Manager screens, include small information buttons  that you can click to display a small pop-up that describes the components.

About the Cluster Parameters

You'll notice several sliders that you can adjust to modify the configuration of your cluster. As you move these sliders, you'll see how the estimated monthly costs for your cluster change. Here are explanations of the adjustments you can make to your cluster provisioning:

NOTE: Note that you can come back and modify your cluster configuration in the future, so you're not stuck forever with your initial settings.

	Cluster Name	Supply whatever name you want for your Splice Machine cluster.
	Region	You can select in which AWS region your cluster will reside by clicking the previously selected region name, which drops down a list of choices.
Data Sizing	Internal Dataset (TB)	<p>Move the slider to modify your estimate of how large your database will be.</p> <p>Internal Dataset is the amount of data that you will be storing within your Splice Machine database.</p>
	External Dataset (TB)	<p>Move the slider to modify your estimate of how large your external dataset will be.</p> <p>External Dataset is the amount of data the you will be accessing from external data sources, using features such as external tables and our virtual table interface.</p>

Cluster Power	OLTP Splice Units	Move the slider to modify your estimate of how much processing power you need for transactional query processing. More OLTP units means more region servers in your cluster.
	OLAP Splice Units	Move the slider to modify your estimate of how much processing power you need for analytical query processing. More OLAP units means more Spark executors.
Backup Frequency	Frequency	<p>Select how frequently you want Splice Machine to back up your database. You can select Hourly, Daily, or Weekly; each selection displays additional backup timing and retention options:</p> <p>Hourly:</p>  <p>Daily:</p>  <p>Weekly:</p> 



A *Splice Unit* is a measure of processing work; one unit currently translates (approximately) to 2 virtual CPUs and 16 GB of memory.

Modifying Cluster Parameters

We recommend that you spend a few minutes experimenting with modifying the cluster parameters; you'll notice that as you increase various values, the estimated monthly cost of your cluster changes.

When you're satisfied with your cluster configuration parameters, click the **Next** button to set up access to your cluster.

You'll notice that when you increase some values, Splice Machine may indicate that the current setting for a parameter clashes with a change that you've made. For example, in the following image, we have increased the **Internal Dataset** size to 20 TB, and as a result the **Cluster Power** values are no longer adequate to support that large a dataset, as indicated by the striping:

The screenshot shows the 'Create New Cluster' interface. On the left, there are four main configuration sections: **Data Sizing**, **Cluster Power**, **Backup Frequency**, and **Estimated Benchmarks** and **Estimated Costs (Monthly)**.

- Data Sizing:** Shows an 'Internal Dataset (TB)' slider set at 20.1 TB and an 'External Dataset (TB)' slider set at 0 TB.
- Cluster Power:** Shows 'OLTP Splice Units' set at 4 and 'OLAP Splice Units' set at 4. Both values are highlighted with red boxes.
- Backup Frequency:** Set to 'Daily' with a start window of '02:30' and 'Backups to Keep' set to 1.
- Estimated Benchmarks:** Displays performance metrics: Single Record Lookup (ms) at 20ms, Import Rate (record/sec) at 40,000, TPC-C 100 Users (TpmC) at 4,000, and TPC-H Query #2 (sec) at 25.
- Estimated Costs (Monthly):** A donut chart shows a total monthly cost of \$20,623.00, broken down into Storage (\$17,065.00) and Compute (\$3,558.00). A note states: '* Does not include monthly I/O costs.'

At the bottom right are buttons for **Reset**, **Back**, and **Next >**.

Splice Machine will not allow you to create your cluster if any of your values clash. You can click the vertical bar at the end of the striping to instantly set the parameter to the required value.

NOTE: If you don't correct the required setting and attempt to advance to the **Next** screen, you'll see an error message and will be unable to advance until you do correct it.

Configure Cluster Access

Once you've configured your cluster, click the **Next** button to display the **Cluster Access** screen. The following image includes displays of the pop-up help information displays for the different access methods:

The screenshot shows the 'Create New Cluster' interface. On the left, there are two main sections: 'VPC Setup' and 'IAM S3 Access'. 'VPC Setup' includes fields for 'Client VPC connectivity required', 'Account ID', and 'VPC ID (Acceptor)'. 'IAM S3 Access' includes fields for 'S3 connectivity', 'S3 Access Key', and 'S3 Secret Key'. To the right is a 'Virtual Private Cloud Configuration' box with instructions for entering VPC information. Below it is an 'AWS S3 Configuration' box with instructions for IAM privileges. To the far right is a 'Configuration Summary' section listing resources like Internal Dataset (TB), External Dataset (TB), OLTP Splice Units, OLAP Splice Units, Backups, Region, and a Splice Machine Authorization Code. At the bottom is a 'Confirmation' box containing a checked checkbox for accepting terms and conditions. A note below the confirmation box states that an email will be sent once the cluster/database is created.

Virtual Private Cloud Configuration

If applicable, enter your VPC information below to allow the cluster to connect. If you don't have this information ready now, you can always add it later.

AWS S3 Configuration

Important: follow [instructions](#) to set up IAM privileges for Splice to read from and write to select S3 folders. If you don't have this information ready now, you can always add it later.

Configuration Summary

Internal Dataset (TB)	0.1
External Dataset (TB)	0
OLTP Splice Units	4
OLAP Splice Units	4
Backups	daily
Region	us-east-1
Splice Machine Authorization Code	Contact us at websales@splicemachine.com to see if you qualify for a Splice Authorization Code.

Confirmation

I read and accept the [terms and conditions](#).

* An email will be sent to you once cluster/database creation is complete. This email will include instructions for how to connect to your database.

You can set your cluster up for access to your Amazon Virtual Private Cloud (VPC) access by selecting the **Client VPC connectivity required** option and providing your VPC account ID.

You need to configure AWS Identity and Access Management (IAM) for your cluster to allow Splice Machine to access selected S3 folders; this is described in our Configuring an S3 bucket for Splice Machine Acces tutorial.

For more information about Amazon VPC, see <https://aws.amazon.com/vpc/>.

For more information about Amazon IAM, see <https://aws.amazon.com/iam/>.

After setting up any access methods, please confirm that you accept our terms and conditions, then click the **Launch** button, which will take you to the **Payment** screen, unless you've subscribed to Splice Machine from the Amazon Marketplace or have already set up a payment method for your account.

Set Up Payment

When you click the **Launch button**, then one of these actions happens:

- » If you subscribed to Splice Machine via the AWS Marketplace, or you already have a payment method set up on your account, you'll land on your dashboard and will be notified when your cluster has been initialized.
- » If you don't yet have a payment method set up, you'll land on the **Payment** screen, in which you can elect to use one of three payment methods:

Credit Card ACH Authorization Code

Name on Card

Credit Card Number
 4242 4242 4242 4242 MM / YY CVC

Cancel **Submit** >

Credit Card

Credit Card ACH Authorization Code

Name on Card

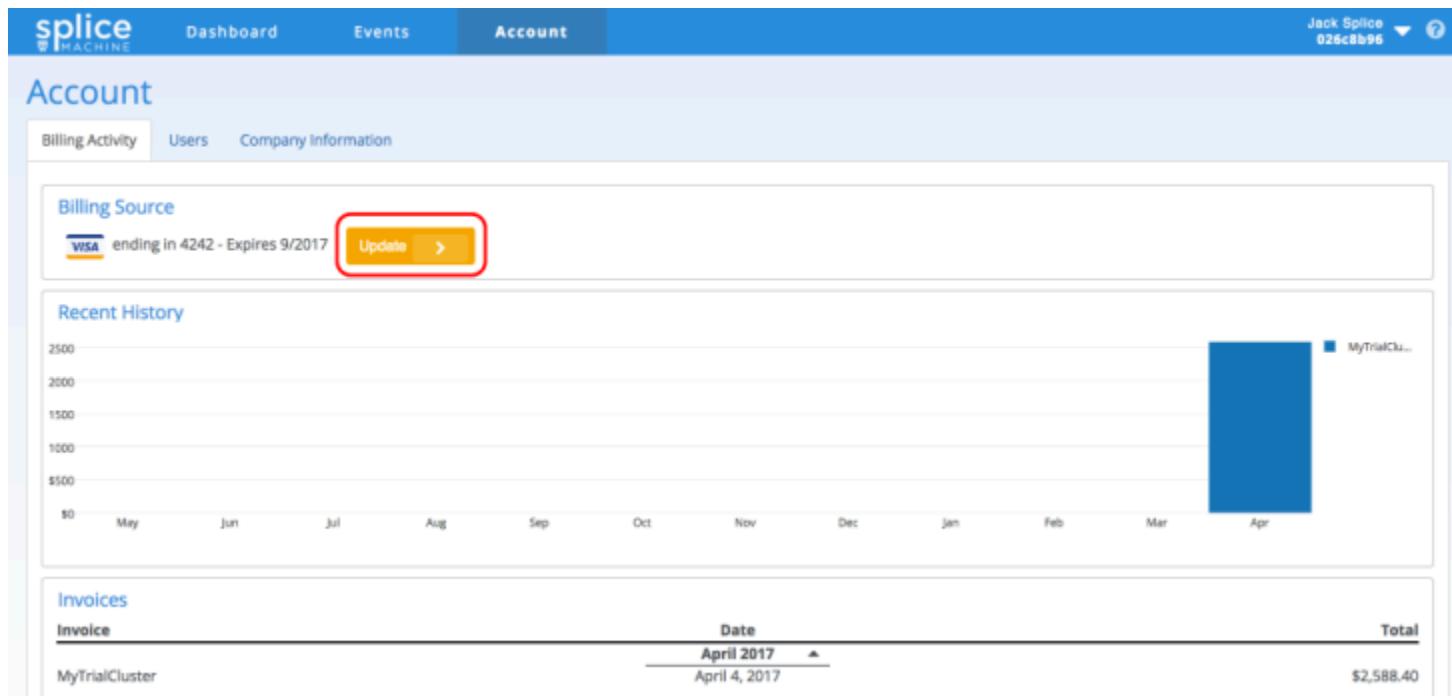
Credit Card Number
 4242 5678 9012 3456 09 / 17 999 99999

Cancel **Submit** >

ACH Electronic Transfer	<p>Credit Card ACH Authorization Code</p> <p>TODO: Implement</p> <p>Cancel Submit ></p>
Authorization Code	<p>Credit Card ACH Authorization Code</p> <p>Authorization Code:</p> <p>3345-8888-9078-ABCF-4321</p> <p>Cancel Submit ></p>

Modifying Payment Information

If you ever need to change your Splice Machine payment information, you can update it in the **Billing Activity** tab of the **Account** screen; just click the **Update** button to revisit the **Payment** screen:



The screenshot shows the Splice Machine web interface with the following details:

- Header:** Splice MACHINE, Dashboard, Events, Account (selected), Jack Splice, 026c8b96, Help.
- Section:** Account
- Billing Activity Tab:** Selected (highlighted in blue).
- Billing Source:** Visa ending in 4242 - Expires 9/2017. An **Update >** button is highlighted with a red box.
- Recent History:** A chart showing revenue over time. The Y-axis ranges from \$0 to \$2500. The X-axis shows months from May to April. A large blue bar represents April revenue, which is partially cut off on the right.
- Invoices:**

Invoice	Date	Total
MyTrialCluster	April 2017 April 4, 2017	\$2,588.40

NOTE: If you've purchased Splice Machine through Amazon Marketplace, change your billing credentials in the Marketplace instead.

Start Using Your Database!

After your cluster spins up, which typically requires about 10 minutes, you can load your data into your Splice Machine database and start running queries.

The easiest way to get going with your new database is to use our Zeppelin Notebook interface, with which you can quickly run queries and generate different visualizations of your results, all without writing any code. We've provided a number of useful Zeppelin tutorials, including one that walks you through setting up a schema, creating tables, loading data, and then running queries.

Note that your data must be in an AWS S3 bucket before you can import it into your Splice Machine database:

- » If you don't yet know how to create an S3 bucket or upload data to a bucket, please check our [Uploading Data to an S3 Bucket](#) tutorial.
- » You may need to configure IAM permissions to allow Splice Machine to access your bucket; see our [Configuring an S3 Bucket for Splice Machine Access](#) tutorial.
- » Once you've got your data in a bucket, you can follow our [Importing Data Tutorial](#) to load that data into Splice Machine.

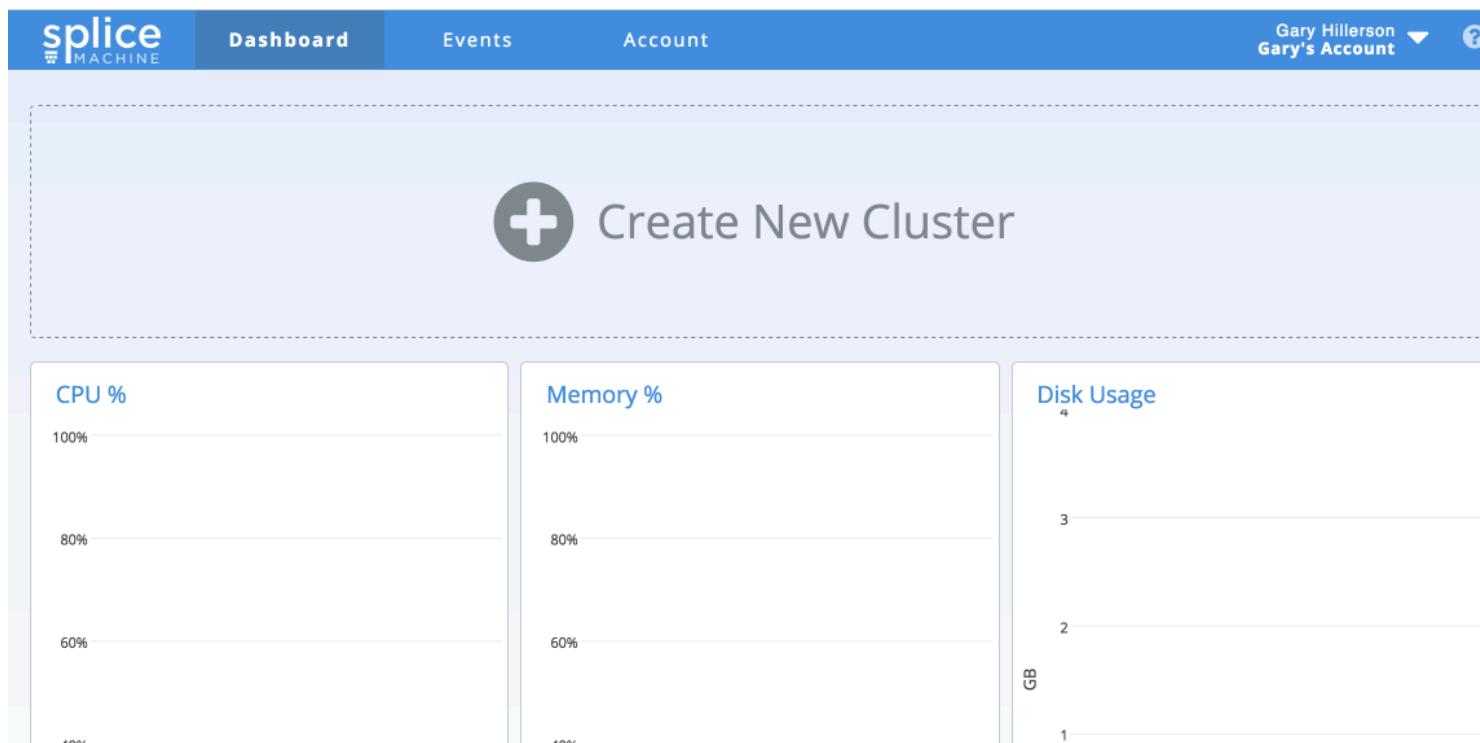
Exploring Your Cloud Manager Dashboard

This topic describes the actions you can initiate from your Splice Machine dashboard, which include:

- » [Creating a New Cluster](#)
- » [Viewing and Managing Your Clusters](#)

Creating a New Cluster

Click the large **Create New Cluster** button to start the process of creating a new cluster. This process, which requires just a few minutes, is described in detail in our Creating a New Cluster topic.



Viewing and Managing Your Clusters

When you create a new cluster, you land back on this Dashboard screen, which shows you the status of your current clusters. In the following image, we've created our first Splice Machine cluster, which is currently *Initializing*.

The screenshot shows the Splice Machine Cloud Manager Dashboard. At the top, there are three tabs: 'Dashboard' (selected), 'Events', and 'Account'. In the top right corner, it says 'Gary Hillerson Gary's Account' with a dropdown arrow and a question mark icon. Below the tabs, the word 'Clusters' is highlighted in blue. To the right of 'Clusters' are 'Refresh' (with a circular arrow icon) and 'Create Cluster' (with a plus sign icon). The main area displays a table of clusters:

Cluster Name	OLTP (Splice units)	OLAP (Splice units)	Storage (TB)	Creation Date	Status
SpliceDocs1	4	4	0.1	6/14/2017 - 17:29	Initializing

Once your new cluster is initialized, its status changes to *Active*, and you receive an email message from Splice Machine notifying you that your cluster is ready. At that point, you can click the cluster name (e.g. *SpliceDocs1*) to use the *Cluster Management* screen for that cluster.

NOTE: If you have multiple clusters associated with your account, each will be listed in your dashboard. Simply click or tap a cluster name to jump to its management screen.

Managing a Cluster

This topic describes the actions you can initiate from the **Cluster Management** screen, which include:

- » [Viewing the cluster's CPU, Disk, and Memory Usage](#)
- » [Reconfiguring cluster access](#)
- » [Resizing your cluster](#)
- » [Deleting your cluster](#)
- » Using Zeppelin and the Database Console with your cluster

The Cluster Management Screen

You can access the **Cluster Management** screen for any cluster in your Dashboard by simply clicking the name of the cluster.

This screen displays information about the cluster, and includes three panels that display a graph of resource usage over time. There are three similar resource usage graphs displayed:

- » CPU Percentage Used (shown in the following image)
- » Disk Usage
- » Memory Usage

SpliceMachine Dashboard Events Account Gary Hillerson Gary's Account ?

SpliceDocs

Reconfigure Resize Delete

OLTP	OLAP	Storage	Creation Date	Status
4 splice units	4 splice units	0.1 TB	6/14/2017 - 18:49	Active

Activity Events Log Details

CPU %

The chart displays CPU usage percentage on the Y-axis (0% to 100%) against time on the X-axis (Jun 14, 2017). Two data series are shown: Spark (blue line) and Region (orange line). Both series show fluctuating activity throughout the day, with a notable peak around 15:25 and another around 15:30.

Live | 1hr | 3hr | 12hr | 1dy | 5dy | 2wk

100%
80%
60%
40%
20%
0%

15:25 Jun 14, 2017 15:30 15:35

Spark
Region

Modifying Your Cluster

Your dashboard features three cluster modification choices; you can Reconfiguring, Resizing, or Deleting your cluster by clicking one of the buttons near the top-left of your Dashboard screen:

SpliceMachine Dashboard Events Account Gary Hillerson Gary's Account ?

SpliceDocs

Reconfigure Resize Delete

DB Console > Notebook >

Reconfiguring Your Cluster

To reconfigure your cluster, click the **Reconfigure** button; the cluster reconfiguration screen displays:

Reconfigure Cluster SpliceDocs2

VPC Setup i

Client VPC connectivity required

Account ID:

VPC ID (Acceptor):

IAM S3 Access i

S3 connectivity

S3 Access Key:

S3 Secret Key:

Cancel Reconfigure >

You can modify your VPC and/or IAM configuration information in this screen. Once you've entered your new information, click the **Reconfigure** button to update your configuration and return to your Cluster Management screen.

Resizing Your Cluster

To resize your cluster, click the **Resize** button. The Resize Cluster screen, which is pretty much identical to the Create New Cluster screen, displays:



Dashboard

Events

Account

Resize SpliceDocs914

Data Sizing

Internal Dataset (TB)

External Dataset (TB)

Cluster Power

OLTP Splice Units

OLAP Splice Units

Backup Frequency

Frequency: Daily

Start Window (UTC): 02:30:00

Backups to Keep: 1

Estimated Benchmarks

Single Record Lookup (ms)	20ms
Import Rate (record/sec)	40,000
TPC-C 100 Users (TpmC)	4,000
TPC-H Query #2 (sec)	25

Estimated Costs (Hourly)

Storage	\$0.09
Compute	\$4.00

Total **\$4.09**

* Does not include monthly I/O costs.

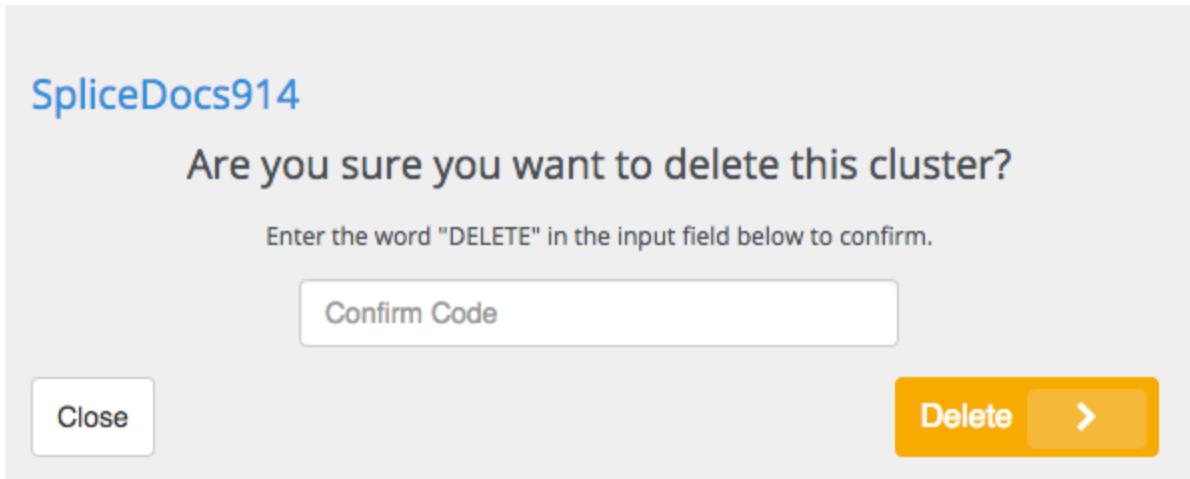
Reset Back **Resize**

Note - resizing a cluster can take some time. We will notify you by email once your cluster is resized.

You can adjust your cluster parameters, then click the **Resize** button, which will return you to your Cluster Management screen, where you'll see the status of your cluster set to *Updating*.

Deleting Your Cluster

To delete your cluster, click the **Delete** button. You'll be asked to confirm the deletion:



After confirming that you want to do so, your cluster will be deleted, and its status in your dashboard will show as *Deleted*.

Working With Your Database

Your dashboard includes buttons with which you can access two different means of working with your database; you can launch the **DB Console** or the Apache Zeppelin **Notebook** interface by clicking one of the buttons near the upper-right corner of the Dashboard screen:

Click the **DB Console** button to land on our Database Console interface.

Click the **Notebook** button to land on our Zeppelin Notebook interface.

Connecting to Your Database with JDBC

At the very bottom of the Cluster Management screen, below the three graph panels, is the URL you can use for a JDBC connection with your database service:

[External Spark Console](#)

JDBC URL: `jdbc:splice://garysaccount-spicedocs.splicemachine:1527/spicedb`

© 2017 Splice Machine, Inc. All rights reserved.

[Terms](#) [Privacy](#) [Trademarks](#)



The same JDBC link was emailed to you in the message from Splice Machine letting you know that your cluster creation was successful.

Managing Your Splice Machine Account

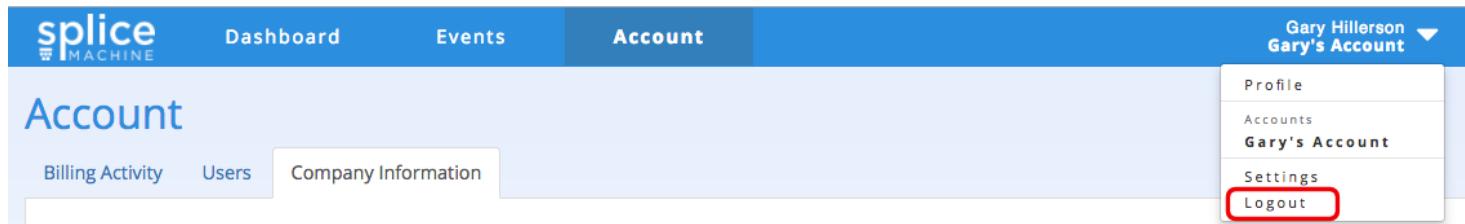
This is a DB-Service-Only topic! [Learn about our products](#)

This topic describes the actions you can perform from the Account tab and Account drop-down in your Dashboard, which include:

- » [Logging Out of Your Account](#)
- » [Reviewing and Updating Your Billing Information](#)
- » [Viewing and Updating Your User Profile and Password](#)
- » [Viewing and Adding Users](#)
- » [Reviewing and Updating Your Company Information](#)

Logging Out of Your Account

To log out of your Cloud Manager account, click the Account Drop-down arrow in the upper-right of your dashboard screen, and select **Logout**:



You'll be logged out and will land back on the Splice Machine Cloud Manager [Login](#) page.

Reviewing and Updating Your Billing Information



If you subscribed to Splice Machine via the AWS Marketplace, your billing is handled by AWS, not Splice Machine. Your *Account Management* screen will not contain a **Billing Activity** tab; this section does not apply to you.

To display billing information for your account, select the **Billing Activity** tab in a Cloud Manager screen. You can see billing details for each month of each year that your account has been alive. You can also hover over one of the bars representing a cluster to see exactly how much that cluster cost in a month (as shown for July in the image below).

If you have provisioned more than one cluster in your account, each cluster is shown in a different color in the billing detail graphic, as shown below.

The screenshot shows the Splice Machine Account page. At the top, there are tabs for Dashboard, Events, and Account (which is selected). On the right, it shows a user profile for "Jack Splice" with the ID "026c8b96". Below the tabs, there are three main sections: "Billing Activity" (selected), "Users", and "Company Information".

- Billing Source:** Shows a VISA card ending in 4242 - Expires 9/2017. An orange "Update" button with a right-pointing arrow is highlighted with a red box.
- Recent History:** A chart showing monthly spending from May to April. The Y-axis ranges from \$0 to \$2500. The X-axis lists months from May to April. A single blue bar represents "MyTrialCluster" for April, reaching approximately \$2,588.40.
- Invoices:** A table showing invoices for "MyTrialCluster" dated April 2017, totaling \$2,588.40.

Invoice	Date	Total
MyTrialCluster	April 2017 April 4, 2017	\$2,588.40

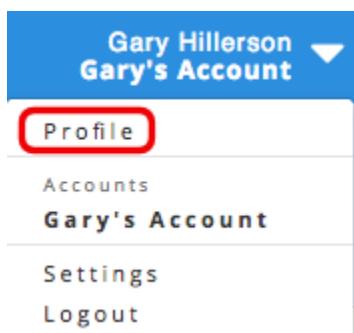
To update your payment source, click the **Update** button.

Prorated Monthly Billing

Splice Machine bills for our database service on a prorated monthly basis; any adjustments for deleting or downsizing your cluster(s) are applied to future bills or cluster purchases.

Viewing and Updating Your User Profile and Password

You can review or edit your profile information by selecting **Profile** from the click the Account Drop-down:



The **Profile** screen displays:

The screenshot shows two main sections of the Splice Machine Cloud Manager interface:

- Profile Info**: This section displays the user's name information. It includes fields for **First Name** (Gary) and **Last Name** (Hillerson). Below these fields is an orange **EDIT** button followed by a right-pointing arrow.
- Reset Password**: This section contains a text input field labeled "Enter Email" for entering an email address to receive a password reset link. Below the input field is an orange **Reset Password** button followed by a right-pointing arrow.

You can edit your name information by clicking the **EDIT** button in the Profile Info panel.

You can reset your account password by entering your email address and then clicking the **Reset Password** button. You'll receive an email from Splice Machine that contains a link you can use to reset your password.

Viewing and Adding Users

To display the names and log-in information for the users of your database service, select the **Users** tab in your Cloud Manager screen. The **Users** screen displays:

Billing Activity Users Company Information

Users

User Name	Email	Last Login
Gary Hillerson	ghillerson@splicemachine.com	6/14/2017

Invite User +

To add another user, click the **Invite User +** button in the *Users* screen. Then enter the new user's email address in the *Invite User* screen and click the **Send** button. We'll send an email inviting that person to set up a password to access your database.

Invite User

Email

Email

* An email will be sent to this address to set up a password

Cancel **Send >**

Reviewing and Updating Your Company Information

To display the company information associated with your account, select the **Users** tab in your Cloud Manager screen. The *Company Information* screen displays:

The screenshot shows the Splice Machine account interface. At the top, there's a navigation bar with tabs for Dashboard, Events, and Account. The Account tab is selected. On the far right of the top bar, it says "Gary Hillerson" and "Gary's Account". Below the navigation bar, there are three tabs: Billing Activity, Users, and Company Information, with Company Information being the active tab. The main content area has a title "Company Info" and a table of company details. At the bottom right of the table is an orange "EDIT" button with a right-pointing arrow.

Account Name	Gary's Account
Admin Email Address	ghillerson@splicemachine.com
Street Address	-
Street Address 2	-
City	-
State	-
Zip Code	-
Phone Number	-
Country	-

EDIT >

To edit the company information associated click the **Edit** button.

Managing Your Event Notifications

This is a DB-Service-Only topic! [Learn about our products](#)

This topic describes the Splice Machine Events Manager, which allows you to examine notification messages sent to your cluster.

Here's a screenshot of a partially populated **Events Manager screen**:

The screenshot shows the Splice Machine Events Manager interface. At the top, there is a navigation bar with tabs for Dashboard, Events (which is selected), and Account. On the far right, it shows the user's name, "Gary Hillerson", and account name, "Gary's Account". Below the navigation bar, the title "Events" is displayed. Underneath the title, there is a search/filter section with fields for "Cluster" (set to "GaryDocs2"), "Start Date" (button to "Choose a date"), "End Date" (button to "Choose a date"), and "Keyword Search" (empty input field). To the right of these fields are "Filter" and "Clear Filter" buttons. Below the filter section is a table listing events. The table has two columns: "Date" and "Detail". The "Date" column lists specific times on 7/25/2017, and the "Detail" column provides a brief description of each event, such as "Daily backup completed at 2017/07/18 12:35:02AM". A pagination control with buttons for «, <, 1, >, » is located above the event list. The event list itself contains six entries, each with a small downward arrow icon to its right.

Date	Detail
7/25/2017 - 20:07:45	Daily backup completed at 2017/07/18 12:35:02AM
7/25/2017 - 20:07:21	Daily backup is scheduled for tonight at 2017/07/18 12:00:00AM
7/25/2017 - 20:07:03	Daily backup completed at 2017/07/17 12:33:59AM
7/25/2017 - 20:06:38	Daily backup is scheduled for tonight at 2017/07/17 12:00:00AM
7/25/2017 - 20:06:24	Daily backup completed at 2017/07/16 12:34:51AM
7/25/2017 - 20:06:11	Daily backup is scheduled for tonight at 2017/07/16 12:00:00AM

You can initiate these actions in the Events Manager:

- » Display messages for one specific cluster, or all of your clusters; in the screenshot above, events are displayed for the cluster named *GaryDocs2*.
- » Filter which messages are displayed; enter filter criteria, then click the **Filter** button. You can filter on:
 - » A start date.
 - » An end date.
 - » A keyword or exact phrase.

You can filter on a start date or end date on its own, or combine them together to specify a date range. You can also combine a date or date-range filter with a keyword filter to find only events that meet the combined criteria.

- » You can click the **Clear Filter** button to clear any filters and display all of your notification messages.

- » Click the < (Prev), > (Next), << (First), or >> (Last) buttons to move through multiple screenfuls of messages.
- » Click the arrow to the right of a message to display the full or shortened version of the message.

Using Zeppelin Notebooks

This is a DB-Service-Only topic! [Learn about our products](#)

This guide helps you to get started with using Zeppelin notebooks to interact with your Splice Machine database.

Topic	Description
<i>Getting Started with Zeppelin</i>	Introduces you to using Zeppelin with your Splice Machine database.
<i>Zeppelin Usage Notes</i>	Specific usage notes for creating Zeppelin notebooks to use with Splice Machine..
<i>A Simple Tutorial</i>	Walks you through a quick and simple tutorial that shows you how to use Zeppelin notebooks to load and query data, and apply different visualizations to the results.

Getting Started with Zeppelin

This is a DB-Service-Only topic! [Learn about our products](#)

This topic helps you to get started with using Zeppelin with your Splice Machine database.

NOTE: We strongly encourage you to visit the [Zeppelin documentation site](#) to learn about creating, modifying, and running your own Zeppelin notebooks.

The Zeppelin Dashboard

When you click the **Notebook** button in your Cluster Management dashboard, you land on the Zeppelin welcome page. To start using Zeppelin with your database service, you need to log in to your database by clicking the **Login** button.

Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Help

Get started with [Zeppelin documentation](#)

Community

Please feel free to help us to improve Zeppelin,
Any contribution are welcome!

[Mailing list](#)
 [Issues tracking](#)
 [Github](#)

Use the same user ID and password to log into Zeppelin as you use to log into your database.

When you log into Zeppelin for your database, you'll land on the Zeppelin dashboard, which displays the list of available notebooks. As you can see, notebooks can be organized in folders.

Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics.

You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook ▾

Import note

Create new note

Filter

ETL Pipeline Example

IoT Demo

1.Overview

2.Database Setup

3.Kafka

4.SparkStream

5.Splice Query

Splice Tutorials

Simple Example

Supply Chain IoT - Crystal Ball

TPCH-1

Utilities

TimelineWithSimulator

Zeppelin Tutorial

Basic Features (Spark)

R (SparkR)

Using Mahout

Help

Get started with [Zeppelin documentation](#)

Community

Please feel free to help us to improve Zeppelin,
Any contribution are welcome!



Mailing list

Issues tracking

Github

Splice Machine has already created a number of useful notebooks; we suggest that you try running some of them to get a feel for what Zeppelin can do: click a notebook name, and you'll land on the notebook page in Zeppelin. From there, you can run all or portions of the notebook, modify its content, and create new notebooks. Our next topic, A Simple Tutorial, uses the our [Simple Example](#) tutorial.

First Notebook Run: Save Interpreter Bindings

The first time that you run any Zeppelin notebook, you need to bind any interpreters needed by the notebook. For our tutorials, these are preconfigured for you; all you need to do is click the Save button:

The screenshot shows the Splice Machine Zeppelin interface. At the top, there's a blue header bar with the Splice Machine logo, the word "Zeppelin", and navigation links for "Notebook" and "Job". On the right side of the header is a search bar labeled "Search your Notes" with a magnifying glass icon, and a user dropdown menu showing "admin". Below the header, the main content area has a title "Splice Tutorials / Simple Ex..." followed by a toolbar with various icons. The main section is titled "Settings" and contains a heading "Interpreter binding". It explains how to bind/unbind interpreters and lists several interpreters: spark (%spark), md (%md), angular (%angular), python (%python), plsql (%plsql), and splicemachine (%splicemachine). Each interpreter entry includes a circular icon with a refresh symbol. At the bottom of this section are "Save" and "Cancel" buttons. A note at the bottom of the page states: "NOTE: If you neglect to save its bindings, the notebook will not run. And again: you only need to do this one time for each notebook that you run."

The Zeppelin Note Toolbar

Zeppelin displays a toolbar at the top of each note that provides convenient access to a number of options:



The following table describes the toolbar buttons:

	Executes all of the paragraphs in the note, in display-order sequence.
--	--

	Shows or hides the code sections of the paragraphs in the note.
	Shows or hides the result sections of the paragraphs in the note.
	Clears the result sections of the paragraphs in the note.
	Clones the current note.
	Exports the current note in JSON format. NOTE: The code and result sections of all paragraphs are exported; you might want to clear your results before exporting a note.
	Switches between personal and collaboration modes.
	Commits changes that you've made to the content of the current note (and allows you to add a commit note).
Head ▾	Displays the revision you're currently viewing, and lets you select from available revisions.
	Deletes the note.
	Schedules execution of the note, using CRON syntax.

The Zeppelin Drop-Down Menu

When you're working in Zeppelin, you can quickly jump to another notebook or create a new note by clicking the **Zeppelin** drop-down menu:

The screenshot shows the Splice Machine Zeppelin interface. At the top, there's a blue header bar with the Splice Machine logo and the word "Zeppelin". Below the header, there are two tabs: "Notebook" and "Job". The "Job" tab is currently active. To the right of the tabs is a sidebar. At the top of the sidebar is a button labeled "+ Create new note". Below it is a search bar with a magnifying glass icon labeled "Filter". Underneath the search bar is a list of job names, each preceded by a small icon: "ETL Pipeline Example", "IoT Demo", "Splice Tutorials", "TimelineWithSimulator", and "Zeppelin Tutorial".

Monitoring Job Status

You can monitor the status of any Zeppelin notebook job(s) running in your cluster by clicking the **Job** button at the top of the Zeppelin screen. This displays a list of the notebook jobs that are running and have run on your cluster.

The screenshot shows the "Job" screen of the Splice Machine Zeppelin interface. At the top, there's a blue header bar with the Splice Machine logo and the word "Zeppelin". Below the header, there are two tabs: "Notebook" and "Job". The "Job" tab is active. On the left side, there's a search bar with a placeholder "Search for Job" and a magnifying glass icon. Next to it is a dropdown menu set to "spark". Above the search bar is a small icon representing a list or table. To the right of the search bar is a "Search your Notes" input field with a magnifying glass icon. Further to the right is a user profile icon with the name "splice" and a dropdown arrow. Below the header, the main area is titled "Job". It contains three entries, each representing a running job:

- Splice Tutorials / Simple Example - spark**: Last updated 3 months ago, status READY. Progress bar is mostly green with a few black dots.
- Splice Tutorials / Supply Chain IoT - Crystal Ball - spark**: Last updated 19 days ago, status READY. Progress bar is mostly green with some black and white dots.
- TimelineWithSimulator - spark**: Last updated 18 days ago, status READY. Progress bar is mostly green with a few black dots.

From the **Job** screen, you can:

- » Monitor all jobs associated with your account.
- » Filter which jobs are displayed.
- » Search for notebooks.

- » Start, Pause, or Terminate a running job.
- » Click a notebook job name to navigate to that notebook.

Creating Notebooks

Be sure to view our Usage Notes page for important information about creating Zeppelin notebooks to use with Splice Machine.

Zeppelin Usage Notes

This is a DB-Service-Only topic! [Learn about our products](#)

This page currently contains exactly one tip about using Zeppelin with Splice Machine; this will grow into a loose collection of tips over time.

Use Full Classpath!

If you're coding a Zeppelin notebook in Java, you must specify the full class of imported classes, such as `java.sql.Timestamp`; otherwise, an error occurs.

For example, this generates an error:

```
%spark
import java.util.Date
import java.sql.
{Connection, Timestamp}
classOfTimestamp
classOffoo
val tt = Timestamp.valueOf("2261-12-31 00:00:00")
class foo extends Object { val xx: Timestamp = Timestamp.valueOf("2261-12-31 00:00:00")
}

import java.util.Date
import java.sql.{Connection, Timestamp}
res12: Class[java.sql.Timestamp] = class java.sql.Timestamp
res13: Class[foo] = class foo
tt: java.sql.Timestamp = 2261-12-31 00:00:00.0
<console>:13: error: not found: type Timestamp
val xx: Timestamp = Timestamp.valueOf("2261-12-31 00:00:00")
^
<console>:13: error: not found: value Timestamp
val xx: Timestamp = Timestamp.valueOf("2261-12-31 00:00:00")
^
ERROR
```

The error is resolved by specifying the full classpath:

```
%spark
import java.util.Date
import java.sql.{Connection, Timestamp}
classOfTimestamp
classOffoo
val tt = Timestamp.valueOf("2261-12-31 00:00:00")
class foo extends Object { val xx: java.sql.Timestamp = java.sql.Timestamp.valueOf("226
1-12-31 00:00:00") }

import java.util.Date
import java.sql.{Connection, Timestamp}
res14: Class[java.sql.Timestamp] = class java.sql.Timestamp
res15: Class[foo] = class foo
tt: java.sql.Timestamp = 2261-12-31 00:00:00.0
defined class foo
FINISHED
```

A Simple Zeppelin Tutorial

This is a DB-Service-Only topic! [Learn about our products](#)

This topic walks you through using a very simple Zeppelin notebook, to help you learn about using Zeppelin with Splice Machine.

NOTE: Our Getting Started with Zeppelin page provides a very brief overview of using Zeppelin; If you're new to Zeppelin, we strongly encourage you to visit the [Zeppelin documentation site](#) to learn about creating, modifying, and running your own Zeppelin notebooks.

Running the Tutorial Notebook

You can access this Zeppelin notebook by clicking the Basics (Spark) link under Zeppelin Tutorials on the Zeppelin Dashboard page:

Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics.

You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

Import note

Create new note

Filter

ETL Pipeline Example

IoT Demo

1.Overview

2.Database Setup

3.Kafka

4.SparkStream

5.Splice Query

Splice Tutorials

Simple Example

Supply Chain IoT - Crystal Ball

TPCH-1

Utilities

TimelineWithSimulator

Zeppelin Tutorial

Basic Features (Spark)

R (SparkR)

Using Mahout

Help

Get started with [Zeppelin documentation](#)

Community

Please feel free to help us to improve Zeppelin,
Any contribution are welcome!



Mailing list

Issues tracking

Github

Once you've opened the tutorial, you can run each step (each Zeppelin *paragraph*) by clicking the **Ready** button that you'll see on the right side of each paragraph. This example includes these steps:

- » Click the first **READY** button to create the schema and a table:

Create a schema called EXAMPLE

FINISHED ▶ X ■ ⊞

```
%spliceMachine
CREATE SCHEMA EXAMPLE;
```

Query executed successfully. Affected rows : 0

Took 4 sec. Last updated by splice at September 14 2017, 10:09:30 AM.

Create a table called LINEITEM

FINISHED ▶ X ■ ⊞

```
%spliceMachine
CREATE TABLE EXAMPLE.LINEITEM (
    L_ORDERKEY BIGINT NOT NULL,
    L_PARTKEY INTEGER NOT NULL,
    L_SUPPKEY INTEGER NOT NULL,
    L_LINENUMBER INTEGER NOT NULL,
    L_QUANTITY DECIMAL(15,2),
    L_EXTENDEDPRICE DECIMAL(15,2),
    L_DISCOUNT DECIMAL(15,2),
    L_TAX DECIMAL(15,2),
    L_RETURNFLAG VARCHAR(1),
    L_LINESTATUS VARCHAR(1),
    L_SHIPDATE DATE,
    L_COMMITDATE DATE,
    L_RECEIPTDATE DATE,
    L_SHIPINSTRUCT VARCHAR(25),
    L_SHIPMODE VARCHAR(10),
    L_COMMENT VARCHAR(44),
    PRIMARY KEY(L_ORDERKEY,L_LINENUMBER)
);
```

Query executed successfully. Affected rows : 0

Took 2 sec. Last updated by splice at September 14 2017, 10:09:37 AM.

- » Import data (in this case, TPCH1 benchmark data) into the table, then verify the data load by counting the number of records in the table:

Import Data

READY ▶ X ■ ⊞

```
%spliceMachine
call SYSCS_UTIL.IMPORT_DATA ('EXAMPLE', 'LINEITEM', null, 's3a://splice-benchmark-data/flat/TPCH/1/lineitem', '|', null, null, null, null, 0, '/tmp',
    , true, null);
```

Took 19 sec. Last updated by anonymous at April 24 2017, 6:56:51 AM.

Count the number of records in the table

READY ▶ X ■ ⊞

```
%spliceMachine
SELECT COUNT(*) AS NUM_RECORDS FROM EXAMPLE.LINEITEM;
```

Took 1 sec. Last updated by anonymous at April 24 2017, 6:57:37 AM.

- » Create indexes on the table, and then run compaction on the data, which is always a good idea after updating a large number of records:

Create some indexes

```
|*splicemachine
create index EXAMPLE.L_SHIPDATE_IDX on EXAMPLE.LINEITEM(
    L_SHIPDATE,
    L_PARTKEY,
    L_EXTENDEDPRICE,
    L_DISCOUNT
);
create index EXAMPLE.L_PART_IDX on EXAMPLE.LINEITEM(
    L_PARTKEY,
    L_ORDERKEY,
    L_SUPPKEY,
    L_SHIPDATE,
    L_EXTENDEDPRICE,
    L_DISCOUNT,
    L_QUANTITY,
    L_SHIPMODE,
    L_SHIPINSTRUCT
);
```

Took 1 sec. Last updated by anonymous at April 24 2017, 6:56:52 AM.

READY ▶ ✎ 📈 ⏹

Run compaction

```
|*splicemachine
call SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA('EXAMPLE');
```

Took 2 sec. Last updated by anonymous at April 24 2017, 6:56:54 AM.

READY ▶ ✎ 📈 ⏹

- » Collect statistics, to improve query planning, and then run a query:

Collect Statistics

```
|*splicemachine
analyze schema EXAMPLE;
```

Took 2 sec. Last updated by anonymous at April 24 2017, 6:56:55 AM.

READY ▶ ✎ 📈 ⏹

Run a query

```
|*splicemachine
select
    l_returnflag,
    l_linenstatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    EXAMPLE.lineitem
where
    l_shipdate <= date({fn TIMESTAMPADD(SQL_TSI_DAY, -90, cast('1998-12-01 00:00:00' as timestamp))})
group by
    l_returnflag,
    l_linenstatus
order by
    l_returnflag,
    l_linenstatus
```

Took 1 sec. Last updated by anonymous at April 24 2017, 6:56:55 AM.

READY ▶ ✎ 📈 ⏹

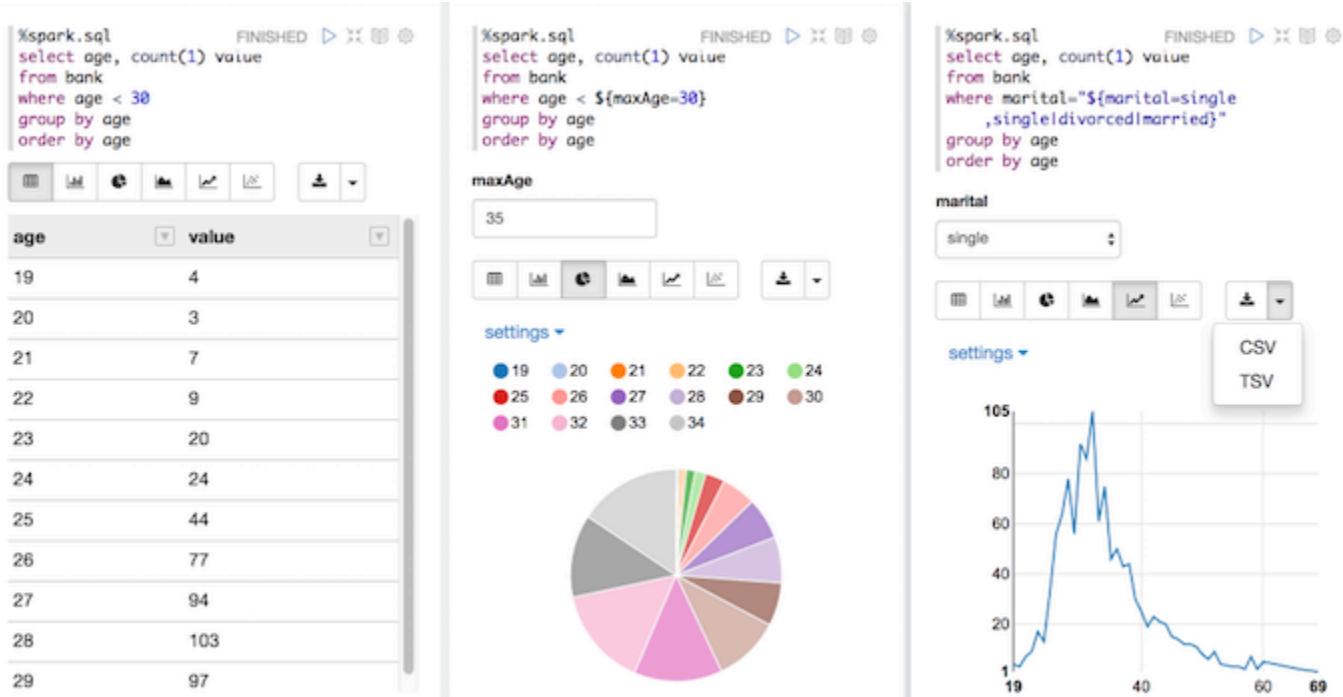
After the query runs, you can take advantage of Zeppelin's built-in visualization tools to display the query results in various graphical and tabular formats.

When you click the **READY** button, Zeppelin runs the paragraph that loads your data and subsequently displays the *Finished* message.

NOTE: If you see *Error* instead of *Finished*, it usually means that you've forgotten to set SpliceMachine interpreter as the default.

Apply Different Visualizations to Your Results

Zeppelin provides a wealth of data visualization tools you can use. In the example below, we have modified the presentation of query results to use different visualizations by clicking different visualization icons in the output pane. You can define and modify the values of variables that you use in your queries; for example, the `maxAge` and `marital` values in the examples below:



Welcome to the Splice Machine On-Premise Database!

Welcome to Splice Machine, the database platform for adaptive applications that manage operational processes. This site contains documentation for our [On-Premise Database](#).



If you're not yet familiar with our lineup of products, please visit the [Getting Started Page](#) in our web site to learn more about them.

Splice Machine delivers an open-source data platform that incorporates the proven scalability of HBase and the in-memory performance of Apache Spark. The cost-based optimizer uses advanced statistics to choose the best compute engine, storage engine, index access, join order and join algorithm for each task. In this way, Splice Machine can concurrently process transactional and analytical workloads at scale.

You can deploy the Splice Machine *On-Premise Database* on a standalone computer or on your own cluster that is managed by Cloudera, MapR, or Hortonworks. We offer Enterprise, Cluster Community, and Standalone Community editions. You can also access the open source code for our community editions on GitHub.

Getting Started

To get started, read about our products on our web site and decide which edition is the right one for you, then download a version or contact our sales team to start using Splice Machine:

- » Learn more about our *Enterprise Edition* features and advantages. To purchase this edition, please [contact Splice Machine Sales](#) today.
- » [Download our free](#) *Cluster Community* edition to install on your own cluster.
- » [Download our free](#) *Standalone Community* edition to run on your MacOS, Linux, or CentOS computer.

Using our Documentation

Please visit our Documentation Summary page for a quick introduction to navigating and using the components of our documentation system.

Version Information

This site contains the documentation for version 2.5 of Splice Machine.

If you're using an earlier version, you'll find the documentation here:

Version 2.5

[Release 2.5 Documentation](#) (Feb 2017)

Splice Machine Requirements

This is an On-Premise-Only topic! [Learn about our products](#)

This topic summarizes the hardware and software requirements for Splice Machine running on a cluster or on a standalone computer.

Cluster Node Requirements

The following table summarizes the minimum requirements for the nodes in your cluster:

Component	Requirements
Cores	Splice Machine recommends that each node in your cluster have 8-12 hyper-threaded cores (16-32 hyper-threads) for optimum throughput and concurrency.
Memory	We recommend that each machine in your cluster have at least 64 GB of available memory.
Disk Space	Your root drive needs to have at least 100 GB of free space. Splice Machine recommends separate data drives on each cluster node to maintain a separation between the operating system and your database data. You need capacity for a minimum of three times the size of the data you intend to load; the typical recommended configuration is 2 TB or more of attached storage per node. Your data disks should be set up with a single partition and formatted with an <code>ext4</code> file system.
Hadoop Ecosystem	The table in the next section, Hadoop Ecosystem Requirements, summarizes the specific Hadoop component versions that we support in each of our product releases.
Software Tools and System Settings	The <i>Linux Configuration</i> topic in each section of our <i>Installation Guide</i> that pertains to your installation summarizes the software tools and system settings required for your cluster machines.

Amazon Web Services (AWS) Requirements

If you're running on AWS, your cluster must meet these minimum requirements:

Component	Requirements
Minimum Cluster Size	The minimum cluster size on AWS is <i>5 nodes</i> : <ul style="list-style-type: none"> • 1 master node • 4 worker nodes
Minimum Node Size	Minimum recommended size of each node is <i>m4.4xlarge</i> .
Disk Space	Minimum recommended storage space: <ul style="list-style-type: none"> • <i>100GB</i> EBS root drive • 4 EBS data drives per node <p>Note that the required number of data drives per node depends on your use case.</p>

Hadoop Ecosystem Requirements

The following table summarizes the required Hadoop ecosystem components for your platform:

Hadoop platform	Linux	Hadoop	HBase	ZooKeeper
CDH 5.13.2, CDH 5.12.0, CDH 5.8.3	CentOS/RHEL 6	2.6.0	1.0.0	3.4.5
HDP 2.6.3, HDP 2.5.5	CentOS/RHEL 6	2.7.1	1.1.2	3.4.5
MapR 5.2.0	CentOS/RHEL 6	2.7.0	1.1.1	3.4.5

Java JDK Requirements

Splice Machine supports the following versions of the Java JDK:

- » Oracle JDK 1.8, update 60 or higher

NOTE: We recommend that you do not use JDK 1.8.0_40

Splice Machine does not test our releases with OpenJDK, so we recommend against using it.

Standalone Version Prerequisites

You can use the standalone version of Splice Machine on MacOS and Linux computers that meet these basic requirements:

Component	Requirements
Operating System	Mac OS X, version 10.8 or later. CentOS 6.4 or equivalent.
CPU	Splice Machine recommends 2 or more multiple-core CPUs.
Memory	At least 16 GB RAM, of which at least 10 GB is available.
Disk Space	At least 100 GB of disk space available for Splice Machine software, plus as much space as will be required for your data; for example, if you have a 1 TB dataset, you need at least 1 TB of available data space.
Software	You must have JDK installed on your computer.

Splice Machine Enterprise Edition Support

The following table summarizes support for the Enterprise Edition of Splice Machine.

This is an On-Premise-Only topic! [Learn about our products](#)

Support Summary	
Term	1 year
Named Support Contacts	5 contacts
Access Channels	Web and Phone
Support Hours	24x7
Support Response Times	
Severity 1	1 hour
Severity 2	4 hours
Severity 3	1 business day
Severity 4	1 business day
Severity Definitions	
Severity 1	Production down or severely degraded to point of being non-functional
Severity 2	Production functional overall but issues reducing performance or functionality
Severity 3	Development or minor production issue
Severity 4	Question or issue that does not impact production

Release Notes for the Splice Machine On-Premise Product

This is an On-Premise-Only topic! [Learn about our products](#)

This topic includes release notes that are specific to the Splice Machine *On-Premise Database* product, in these sections:

- » [Supported Platforms](#)
- » [Enterprise-only Features](#)
- » [Running the Standalone Version](#)

Most of the information about changes in the Splice Machine database that forms the basis of this product are found in the Splice Machine database release notes.

After Updating

After updating to a new release of Splice Machine, you need to update your stored statement metadata by calling these two system procedures:

```
CALL SYSCS_UTIL.SYSCS_UPDATE_METADATA_STORED_STATEMENTS();
CALL SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE();
```

Supported Platforms

The supported platforms for release 2.7 are:

- » Cloudera CDH 5.12.0, 5.8.3
- » MapR 5.2.0
- » HortonWorks HDP 2.5.5, 2.6.3

Enterprise-only Features

Some features only work on the *Enterprise Edition* of Splice Machine; they **do not** work on the Community Edition of Splice Machine. To obtain a license for the Splice Machine *Enterprise Edition*, please [Contact Splice Machine Sales](#) today.

These are the enterprise-only features in our *On-Premise Database*:

- » Backup/Restore
- » LDAP integration
- » Column-level user privileges
- » Kerberos enablement

- » Encryption at rest

Running the Standalone Version

The supported operating systems for the STANDALONE release of Splice Machine are:

- » Mac OS X (10.8 or greater)
- » Centos (6.4 or equivalent)

Splice Machine Installation Guide

This is an On-Premise-Only topic! [Learn about our products](#)

This *Installation Guide* walks you through installing Splice Machine on your cluster, or on computer if you're using the standalone version.

The fastest way to get started with Splice Machine is to set up our sandbox on the Amazon Web Services (AWS) platform on EC2 instances using cloud.splicemachine.com:



See the [Installing Splice Machine on Amazon Web Services](#) topic for step-by-step instructions for setting up the Splice Machine sandbox.

If you want to download and install Splice Machine on your cluster or standalone computer, please read the remainder of this page, which includes these sections:

- » The Cluster Node Requirements section below details the hardware and ecosystem requirements for installing Splice Machine on a cluster or on a standalone computer.
- » The [Configure Linux for Splice Machine](#) section specifies the Linux software that Splice Machine requires.
- » The [Install Splice Machine](#) links to the platform-specific installation and upgrade pages for each version of Splice Machine.

Installing the Splice Machine Sandbox on AWS

The fastest way to get started with Splice Machine is to set up our sandbox on the Amazon Web Services (AWS) platform on EC2 instances using cloud.splicemachine.com.



See the [Installing Splice Machine on Amazon Web Services](#) topic for step-by-step instructions.

Cluster Node Requirements

The following table summarizes the minimum requirements for the nodes in your cluster:

Component	Requirements
Cores	Splice Machine recommends that each node in your cluster have 8-12 hyper-threaded cores (16-32 hyper-threads) for optimum throughput and concurrency.
Memory	We recommend that each machine in your cluster have at least 64 GB of available memory.

Component	Requirements
Disk Space	<p>Your root drive needs to have at least 100 GB of free space.</p> <p>Splice Machine recommends separate data drives on each cluster node to maintain a separation between the operating system and your database data. You need capacity for a minimum of three times the size of the data you intend to load; the typical recommended configuration is 2 TB or more of attached storage per node.</p> <p>Your data disks should be set up with a single partition and formatted with an <code>ext4</code> file system.</p>
Hadoop Ecosystem	The table in the next section, Hadoop Ecosystem Requirements, summarizes the specific Hadoop component versions that we support in each of our product releases.
Software Tools and System Settings	The <i>Linux Configuration</i> topic in each section of our <i>Installation Guide</i> that pertains to your installation summarizes the software tools and system settings required for your cluster machines.

Amazon Web Services (AWS) Requirements

If you're running on AWS, your cluster must meet these minimum requirements:

Component	Requirements
Minimum Cluster Size	<p>The minimum cluster size on AWS is <i>5 nodes</i>:</p> <ul style="list-style-type: none"> • 1 master node • 4 worker nodes
Minimum Node Size	Minimum recommended size of each node is <i>m4.4xlarge</i> .
Disk Space	<p>Minimum recommended storage space:</p> <ul style="list-style-type: none"> • <i>100GB</i> EBS root drive • 4 EBS data drives per node <p>Note that the required number of data drives per node depends on your use case.</p>

Hadoop Ecosystem Requirements

The following table summarizes the required Hadoop ecosystem components for your platform:

Hadoop platform	Linux	Hadoop	HBase	ZooKeeper
CDH 5.13.2, CDH 5.12.0, CDH 5.8.3	CentOS/RHEL 6	2.6.0	1.0.0	3.4.5
HDP 2.6.3, HDP 2.5.5	CentOS/RHEL 6	2.7.1	1.1.2	3.4.5
MapR 5.2.0	CentOS/RHEL 6	2.7.0	1.1.1	3.4.5

Java JDK Requirements

Splice Machine supports the following versions of the Java JDK:

- » Oracle JDK 1.8, update 60 or higher

NOTE: We recommend that you do not use JDK 1.8.0_40

Splice Machine does not test our releases with OpenJDK, so we recommend against using it.

Standalone Version Prerequisites

You can use the standalone version of Splice Machine on MacOS and Linux computers that meet these basic requirements:

Component	Requirements
Operating System	Mac OS X, version 10.8 or later. CentOS 6.4 or equivalent.
CPU	Splice Machine recommends 2 or more multiple-core CPUs.
Memory	At least 16 GB RAM, of which at least 10 GB is available.
Disk Space	At least 100 GB of disk space available for Splice Machine software, plus as much space as will be required for your data; for example, if you have a 1 TB dataset, you need at least 1 TB of available data space.
Software	You must have JDK installed on your computer.

Configure Linux for Splice Machine

The following table summarizes Linux configuration requirements for running Splice Machine on your cluster:

Configuration Step	Description
Configure SSH access:	Configure the user account that you're using for cluster administration for password-free access, to simplify installation.
Configure swappiness:	<pre>echo 'vm.swappiness = 0' >> /etc/sysctl.conf</pre>
If you are using Ubuntu:	<pre>rm /bin/sh ; ln -sf /bin/bash /bin/sh</pre>
If you are using CentOS or RHEL:	<pre>sed -i '/requiretty/ s/^/#/' /etc/sudoers</pre>
Required software:	<p>Verify that the following set of software (or packages) is available on each node in your cluster:</p> <ul style="list-style-type: none"> » curl » Oracle JDK 1.8, update 60 or higher. We recommend against using JDK 1.8.0_40 or OpenJDK. <div data-bbox="633 1130 1334 1245" style="border: 1px solid #ccc; padding: 10px; border-radius: 10px; margin-top: 10px;"> <p>NOTE: Your platform management software may re-install JDK during its own installation process.</p> </div> <ul style="list-style-type: none"> » nscd » ntp » openssh, openssh-clients, and openssh-server » patch » rlwrap » wget

Configuration Step	Description
Additional required software on CentOS or RHEL	<p>If you're running on CENTOS or RHEL, you also need to have this software available on each node:</p> <ul style="list-style-type: none"> » ftp » EPEL repository
Services that must be started	<p>You need to make sure that the following services are enabled and started:</p> <ul style="list-style-type: none"> » nscd » ntpd (ntp package) » sshd (openssh-server package)
Time zone setting	Make sure all nodes in your cluster are set to the same time zone.

Install Splice Machine

If you've decided to try our sandbox, see the [Installing Splice Machine on Amazon Web Services](#) topic for step-by-step instructions for setting up the Splice Machine sandbox.

To install Splice Machine on your cluster or standalone computer, click the link below to see the instructions for your platform; each page walks you through downloading and installing and configuring a specific version of Splice Machine:

Installation instructions for Splice Machine are now being maintained in our GitHub repository; you'll find an `installation.md` document in the `docs` subdirectory of each Splice Machine release/platform directory. For example:

Splice Machine Version	Platform Version	Install Instructions URL
2.7	CDH 5.13.2	https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/cdh5.13.2/docs/CDH-installation.md
	CDH 5.12.0	https://github.com/splicemachine/spliceengine/tree/branch-2.7/platforms/cdh5.12.0/docs/CDH-installation.md
	CDH 5.8.3	https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/cdh5.8.3/docs/CDH-installation.md
	HDP 2.6.3	https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.6.3/docs/HDP-installation.md

Splice Machine Version	Platform Version	Install Instructions URL
	HDP 2.5.5	https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.5.5/docs/HDP-installation.md
	MapR 5.2.0	https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/mapr5.2.0/docs/MapR-installation.md
	Standalone	https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/std/docs/STD-installation.md
2.5	CDH 5.13.2	https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/cdh5.13.2/docs/CDH-installation.md
	CDH 5.12.0	https://github.com/splicemachine/spliceengine/tree/branch-2.5/platforms/cdh5.12.0/docs/CDH-installation.md
	CDH 5.8.3	https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/cdh5.8.3/docs/CDH-installation.md
	HDP 2.6.3	https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/hdp2.6.3/docs/HDP-installation.md
	HDP 2.5.5	https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/hdp2.5.5/docs/HDP-installation.md
	MapR 5.2.0	https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/mapr5.2.0/docs/MapR-installation.md
	Standalone	https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/std/docs/STD-installation.md



For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](#).

Installing Splice Machine on Your Cluster or Computer

This is an On-Premise-Only topic! [Learn about our products](#)

Install Splice Machine

Our installation instructions for Splice Machine are now being maintained in our GitHub repository; you'll find an `installation.md` document in the `docs` subdirectory of each Splice Machine release/platform directory. For example:

Splice Machine Version	Platform Version	Install Instructions URL
2.7	CDH 5.13.2	https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/cdh5.13.2/docs/CDH-installation.md
	CDH 5.12.0	https://github.com/splicemachine/spliceengine/tree/branch-2.7/platforms/cdh5.12.0/docs/CDH-installation.md
	CDH 5.8.3	https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/cdh5.8.3/docs/CDH-installation.md
	HDP 2.6.3	https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.6.3/docs/HDP-installation.md
	HDP 2.5.5	https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.5.5/docs/HDP-installation.md
	MapR 5.2.0	https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/mapr5.2.0/docs/MapR-installation.md
	Standalone	https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/std/docs/STD-installation.md
2.5	CDH 5.13.2	https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/cdh5.13.2/docs/CDH-installation.md
	CDH 5.12.0	https://github.com/splicemachine/spliceengine/tree/branch-2.5/platforms/cdh5.12.0/docs/CDH-installation.md
	CDH 5.8.3	https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/cdh5.8.3/docs/CDH-installation.md
	HDP 2.6.3	https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/hdp2.6.3/docs/HDP-installation.md

Splice Machine Version	Platform Version	Install Instructions URL
	HDP 2.5.5	https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/hdp2.5.5/docs/HDP-installation.md
	MapR 5.2.0	https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/mapr5.2.0/docs/MapR-installation.md
	Standalone	https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/std/docs/STD-installation.md



For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](#).

Installing Splice Machine on Amazon Web Services

This is an On-Premise-Only topic! [Learn about our products](#)

The fastest way to deploy Splice Machine is on the Amazon Web Services (AWS) platform on EC2 instances using cloud.splicemachine.com.



You must have an existing AWS account before running this process.

Getting Started with the SandBox Cluster Generator

To generate your Splice Machine sandbox cluster, point your browser to:

`www.splicemachine.com/get-started/sandbox-start/`

Fill in the form that's displayed:


[CONTACT US](#) [GET STARTED](#)

[Product](#) [Applications](#) [Get Started](#) [Company](#) [Community](#) [Resources](#) [Newsroom](#)
[TRY PRODUCT](#) [DEMO](#)

Try the Splice Machine RDBMS for Yourself

The Splice Machine Community Edition is a fully featured RDBMS, built on Spark and HBase creating a flexible, hybrid database that enables businesses to perform simultaneous OLAP and OLTP workloads.

All versions include features such as:

- ANSI-SQL
- Concurrent ACID Transactions
- OLTP & OLAP Resource Isolation
- Distributed In-Memory Joins, Aggregations, Scans, and Group By's
- Cost-Based Statistics / Query Optimizer
- New Releases and Maintenance Updates

Download Splice Machine through your Amazon E2 account to try Splice Machine's functionality for yourself.

Tutorials, forums, videos and documentation are available through the Splice Machine community at any time at community.splicemachine.com.

If you would like to try the Enterprise Edition, please [contact us](#).

 FIRST NAME *

 LAST NAME *

 EMAIL *

 COMPANY *

 JOB TITLE *

 PHONE *

 COUNTRY*


I ACCEPT THE TERMS OF THE SPLICE MACHINE END
 USER LICENSE AGREEMENT *

Read the Splice Machine [End User License Agreement](#)

[NEXT PAGE](#)


Click the NEXT PAGE button to proceed and configure your cluster.

Configure Your Cluster

AWS provides numerous choices for configuring your sandbox; Splice Machine recommends a minimum configuration such as the following:

SANDBOX CLUSTER GENERATOR

By completing the form below, you can quickly setup a sandbox on Amazon Web Services that will allow you to perform functional testing of Splice Machine v2.0.

You can access the Splice Machine community site at community.splicemachine.com to access tutorials, forums and other resources.

If you would like to try the Enterprise Edition, please [contact us](#).

CloudFormation Stack Name - the name of your new Splice Machine Cluster *

Amount of data you would like to load (GB) *

Please enter a value between **10** and **10000**.

Number of Region Server Nodes *

EC2 KeyPair Name *

This has to be an existing EC2 Keypair in your environment.

EC2 Instance Type *

Pricing is based on estimates and is updated by Amazon on occasion. Please check your account for more details. You will also have a chance to review and approve actual pricing for your cluster on AWS before you launch it.



The EC2 KeyPair Name of your pem file, which can be located anywhere on your computer.

NOTE: When specifying your EC2 KeyPair name, **DO NOT** include any file name suffix. For example, enter `splice-demo`, NOT `splice-demo.pem`.

When you click the SUBMIT button, Splice Machine generates an AWS cluster generator template file, which is a `json` file that you can download for safekeeping.

Thanks for generating your cluster. You can download your file here:

<https://s3.amazonaws.com/splice-cf/20160717213102template-SpliceDoc-demo.json>

Before launching your cluster, select in which AWS region you want your cluster located:

To launch your Cluster select the region you would like to use and then click Launch Cluster:



Click the LAUNCH CLUSTER button to proceed to finalizing your sandbox template.

Finalize Your Sandbox Stack Template

Your sandbox stack template file should already be entered into the Amazon S3 template field at the bottom of the screen; if you're using a different template file, you can select that instead; this is **NOT RECOMMENDED**.

Select Your Sandbox Template

Create stack

Select Template

Specify Details
Options
Review

Select Template

Select the template that describes the stack that you want to create. A stack is a group of related resources that you manage as a single unit.

Design a template Use AWS CloudFormation Designer to create or modify an existing template. [Learn more](#).

Design template

Choose a template A template is a JSON-formatted text file that describes your stack's resources and their properties. [Learn more](#).

Select a sample template

Upload a template to Amazon S3 [Choose File](#) No file chosen

Specify an Amazon S3 template URL <https://s3.amazonaws.com/splice-cf/201607172/> [View/Edit template in Designer](#)

Cancel **Next**

Click the **Next** button to proceed to the *Specify Details* screen.

Rename Your Stack if Desired

You set up a name for your stack at the beginning of this process; however, you can change that here if you want. Then click **Next** to continue on to the *Options* screen.

Create stack

Select Template

Specify Details

Options

Review

Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. [Learn more](#).

Stack name

SpliceDocs-Demo

Cancel

Previous

Next

Specify Options for Your Stack

In the *Options* screen, you can add tags and set advanced options for your sandbox stack.

Adding Resource Tags to the Stack, if Desired

You can optionally tag your sandbox stack with whatever key values you want. For example:

Create stack

Select Template

Specify Details

Options

Review

Options

Tags

You can specify tags (key-value pairs) for resources in your stack. You can add up to 10 unique key-value pairs for each stack. [Learn more](#).

	Key (127 characters maximum)	Value (255 characters maximum)	
1	Department	Engineering	x
2	Owner	SpliceDocs	x
3	Purpose	Splice Machine Sandbox Docs	+

Setting Advanced Options for Your Stack

You can also configure advanced options for your sandbox stack in the *Options* screen, including notification, failure rollback, and policy options. For more information about these options, click the [Learn more](#) button.

▼ Advanced

You can set additional options for your stack, like notification options and a stack policy. [Learn more.](#)

Notification options

No notification

New Amazon SNS topic

Topic	<input type="text"/>
Email	<input type="text"/>

Existing Amazon SNS topic

Existing topic ARN

Timeout  Minutes

Rollback on failure  Yes

No

Stack policy  Enter policy

Upload policy file

No file chosen

[Learn more](#)

Click the **Next** button at the bottom of the *Options* screen to proceed to the *Review* screen.

Review Your Sandbox Stack Configuration

Finally, review your stack configuration, and then click the **Create** button to start creating your sandbox cluster.

Create stack

Select Template

Specify Details

Options

Review

Review

Template

Template URL	https://s3.amazonaws.com/splice-cf/20160717222024template-SpliceDocs-Demo.json
Description	Splice Machine Sandbox Cluster
Estimate cost	Cost

Details

Stack name SpliceDocs-Demo

Create IAM resources No

Options

Tags

Department	Engineering
Owner	SpliceDocs
Purpose	Splice Machine Sandbox Docs

Advanced

Notification

Timeout none

Rollback on failure Yes

[Cancel](#)

[Previous](#)

Create

Finish Launching Your Cluster

When AWS starts creating your cluster, you'll see a progress screen like this:

The screenshot shows the AWS CloudFormation console interface. At the top, there are three buttons: 'Create Stack', 'Actions ▾', and 'Design template'. Below these are two dropdown filters: 'Filter: Active' and 'By Name:'. On the right side, it says 'Showing 5 stacks'. The main area is a table with four columns: 'Stack Name', 'Created Time', 'Status', and 'Description'. A single row is visible for the stack 'SpliceDocs-Demo', which was created on '2016-07-17 16:41:19 UTC-0700' and is currently in the 'CREATE_IN_PROGRESS' state. The description indicates it's a 'Splice Machine Sandbox Cluster'.

	Stack Name	Created Time	Status	Description
<input type="checkbox"/>	SpliceDocs-Demo	2016-07-17 16:41:19 UTC-0700	CREATE_IN_PROGRESS	Splice Machine Sandbox Cluster

Depending on your configuration, the cluster may take some time to create. When it finishes, you'll see that the status has changed to complete:

This screenshot shows the same AWS CloudFormation console interface after the stack has completed creation. The table now displays the stack in the 'CREATE_COMPLETE' state. The 'Status reason' section indicates the cluster is a 'Splice Machine Sandbox Cluster'. The 'Description' field is also present.

	Stack Name	Created Time	Status	Description
<input checked="" type="checkbox"/>	SpliceDocs-Demo	2016-07-17 16:41:19 UTC-0700	CREATE_COMPLETE	Splice Machine Sandbox Cluster

Overview ▾

Stack name: SpliceDocs-Demo
Stack ID: arn:aws:cloudformation:us-east-1:953558963498:stack/SpliceDocs-Demo2/f5c90530-4c77-11e6-9d7a-50d5ca6e604a
Status: CREATE_COMPLETE
Status reason:
Description: Splice Machine Sandbox Cluster

After the status changes to complete, you can display your nodes by clicking the stack name link, and then viewing *Outputs*, which will look something like this:

SpliceDocs-Demo

[Other Actions ▾](#)
[Update Stack](#)
Stack name: SpliceDocs-Demo2

Stack ID: arn:aws:cloudformation:us-east-1:953558963498:stack/SpliceDocs-Demo2/f5c90530-4c77-11e6-9d7a-50d5ca6e604a

Status: CREATE_COMPLETE

Status reason:
Description:

▼ Outputs

Key	Value	Description
manager	http://ec2-52-3-255-134.compute-1.amazonaws.com:7180	URL for Cloudera Manager
worker01	Hostname ec2-54-164-248-229.compute-1.amazonaws.com	splice-worker01 Public DNS
worker04	Hostname ec2-52-91-119-113.compute-1.amazonaws.com	splice-worker04 Public DNS
worker03	Hostname ec2-52-3-255-178.compute-1.amazonaws.com	splice-worker03 Public DNS
worker02	Hostname ec2-52-87-230-155.compute-1.amazonaws.com	splice-worker02 Public DNS
master	Hostname ec2-52-3-255-134.compute-1.amazonaws.com	splice-master Public DNS

Use Cloudera to Manage your Cluster

You can now log into Cloudera Manager for your cluster by clicking the manager link shown in the *Outputs* screen, which lands you on the login screen:

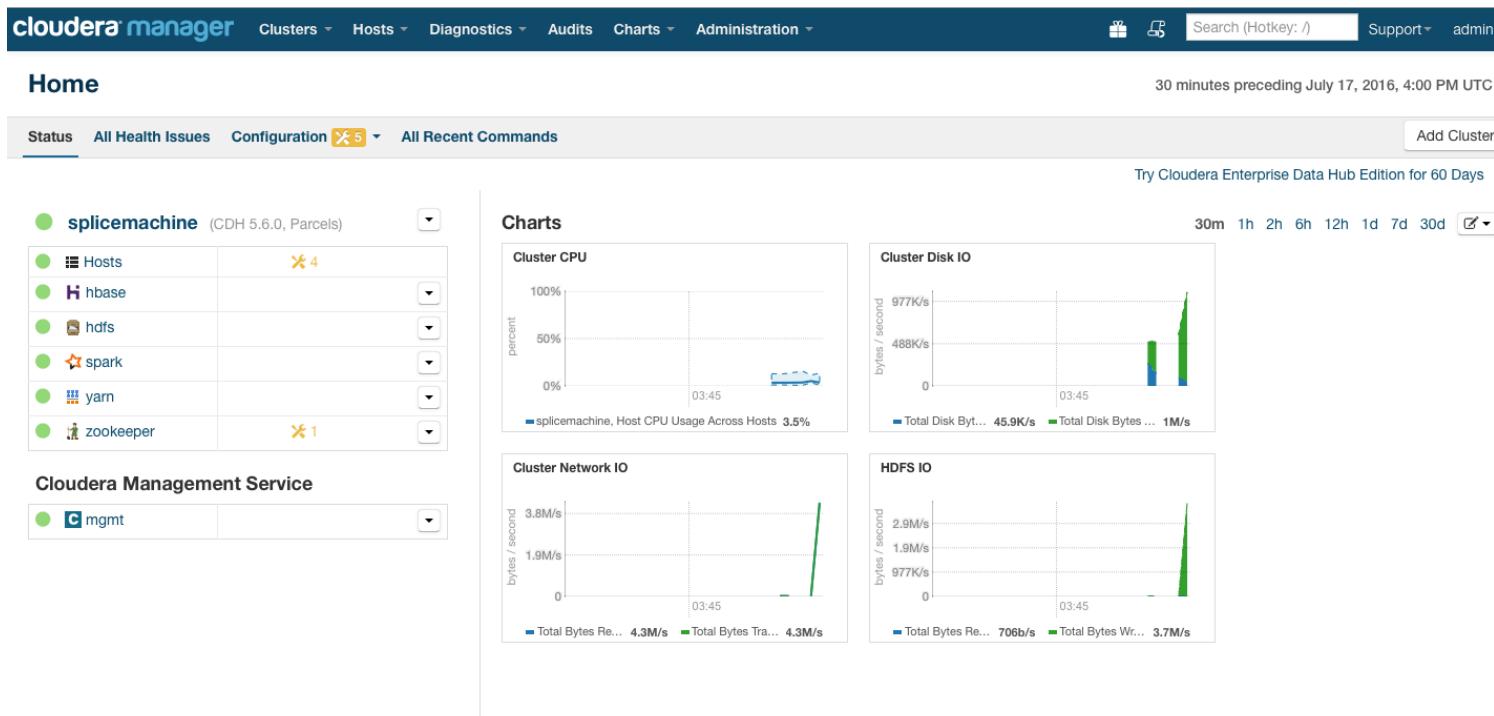
...

...

[Log In](#)

Remember me

After logging in, you can use Cloudera to manage your Splice Machine cluster.



Using Splice Machine with Linux

You can now log into one of your sandbox nodes and use Splice Machine from a terminal window on your computer. To do so, ssh into the address of the node, which you can see in the Outputs screen from when you created the sandbox, or by viewing your cluster hosts in Cloudera Manager.

Make sure that you include the full name of your EC2 KeyPair when connecting. For example:

```
ssh -i ~/Downloads/splice-demo.pem.txt centos@ec2-54-164-248-229.compute-1.amazonaws.com
```

Once you're connected to Splice Machine, you can work with your database. For example:

```
~$ ssh -i ~/Downloads/splice-demo.pem.txt centos@ec2-54-164-248-229.compute-1.amazonaws.com
Last login: Sun Jul 17 23:58:17 2016 from 10.250.0.10
[centos@ip-10-250-0-11 ~]$ sqlshell.sh

===== rlwrap detected and enabled. Use up and down arrow keys to scroll through command line history. =====

Running Splice Machine SQL shell
For help: "splice> help;"
SPLICE* - jdbc:splice://localhost:1527/splicedb
* = current connection
splice> show tables;


| TABLE_SCHEM | TABLE_NAME       | CONGLOM_ID | REMARKS |
|-------------|------------------|------------|---------|
| SYS         | SYSALIASES       | 272        |         |
| SYS         | SYSBACKUP        | 912        |         |
| SYS         | SYSBACKUPFILESET | 1024       |         |
| SYS         | SYSBACKUPITEMS   | 1200       |         |
| SYS         | SYSBACKUPJOBS    | 1232       |         |
| SYS         | SYSCHECKS        | 288        |         |
| SYS         | SYSCOLPERMS      | 688        |         |
| SYS         | SYSCOLUMNS       | 80         |         |
| SYS         | SYSCOLUMNSTATS   | 1280       |         |
| SYS         | SYSCONGLOMERATES | 48         |         |
| SYS         | SYSCONSTRAINTS   | 256        |         |
| SYS         | SYSDEPENDS       | 304        |         |
| SYS         | SYSFILES         | 336        |         |
| SYS         | SYSFOREIGNKEYS   | 352        |         |
| SYS         | SYSKEYS          | 240        |         |
| SYS         | SYSPERMS         | 864        |         |
| SYS         | SYSPHYSICALSTATS | 1296       |         |
| SYS         | SYSPRIMARYKEYS   | 368        |         |
| SYS         | SYSROLES         | 800        |         |
| SYS         | SYSROUTINEPERMS  | 704        |         |
| SYS         | SYSSCHEMAS       | 32         |         |
| SYS         | SYSSEQUENCES     | 816        |         |
| SYS         | SYSSTATEMENTS    | 384        |         |
| SYS         | SYSTABLEPERMS    | 640        |         |
| SYS         | SYSTABLES        | 64         |         |
| SYS         | SYSTABLESTATS    | 1312       |         |
| SYS         | SYSTRIGGERS      | 576        |         |
| SYS         | SYSUSERS         | 880        |         |
| SYS         | SYSVIEWS         | 320        |         |
| SYSIBM      | SYSDUMMY1        | 1328       |         |


30 rows selected
splice>
```

Splice Machine Administrator's Guide

This is an On-Premise-Only topic! [Learn about our products](#)

This *Administrator's Guide* describes the administrative tasks associated with configuring and maintaining your Splice Machine software, in the following topics:

Topic	Description
Starting Your Database	How to start or restart your database.
Backing Up and Restoring	How to back up and restore your Splice Machine database.
Cleaning Your Database	How to clean your Splice Machine database.
Shutting Down Your Database	How to shut down your Splice Machine database.
Accessing Derby Properties	How to enable and disable Derby features by setting Derby properties.
Upgrading to the Enterprise Edition of Splice Machine	Describes how to upgrade from the <i>Community Edition</i> of Splice Machine to the <i>Enterprise Edition</i> .



For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](#).

For a summary of all Splice Machine documentation, see the Documentation Summary topic.

Starting Your Database

This is an On-Premise-Only topic! [Learn about our products](#)

- » [Starting Your Splice Machine Database on a Cloudera-Managed Cluster](#)
- » [Starting Your Splice Machine Database on a Hortonworks HDP-Managed Cluster](#)
- » [Starting Your Splice Machine Database on a MapR-Managed Cluster](#)
- » [Starting Your Splice Machine Database on a Standalone installation](#)

Starting Your Splice Machine Database on a Cloudera-Managed Cluster

Use the Cloudera Manager to start HBase:

1. Navigate to the **Services->All Services** screen in *Cloudera Manager*, and select this action to start **HBase**:

Actions -> Start

Starting Your Splice Machine Database on a Hortonworks HDP-Managed Cluster

Use the Ambari dashboard to start Splice Machine:

1. Log in to the Ambari Dashboard by pointing your browser to the publicly visible <**hostName**> for your master node that is hosting Ambari Server:

`http://<hostName>:8080/`

2. Start cluster services:

Action -> Start All

Starting Your Splice Machine Database on a MapR-Managed Cluster

To start Splice Machine, use the MapR Command System (MCS):

1. Navigate to the node that is running the `mapr-webserver` service.
2. Log into `https://<hostIPAddr>:8443`, substituting the correct `<hostIPAddr>` value. The login credentials are the ones you used when you added the `mapr` user during installation.
3. Start your HBase master
4. Start all of your Region Servers

Starting Your Database in the Standalone Version

Follow these steps to start your database if you're using the Standalone version of Splice Machine:

1. Change directory to your install directory:

```
cd splicemachine
```

2. Run the Splice Machine start-up script:

```
$ ./bin/start-splice.sh
```

Restarting Splice Machine

If you're running the standalone version of Splice Machine on a personal computer that goes into sleep mode, you need to stop and then restart Splice Machine when you wake the computer back up. For example, if you're running Splice Machine on a laptop, and close your laptop, then you'll need to stop and restart Splice Machine when you reopen your laptop.

To stop and restart Splice Machine, follow these steps:

1. Make sure that you have quit the `splice>` command line interpreter:

```
splice> quit;
```

2. Change directory to your install directory:

```
cd splicemachine
```

3. Run the Splice Machine shut-down script:

```
$ ./bin/stop-splice.sh
```

4. Run the Splice Machine start-up

```
$ ./bin/start-splice.sh
```

Backing Up and Restoring Your Database



ENTERPRISE ONLY: This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product.

You cannot use this feature with the Community editions of Splice Machine. For a list of the additional features available in the Enterprise edition, see our Splice Machine Editions page.

To obtain a license for the Splice Machine *Enterprise Edition*, please [Contact Splice Machine Sales](#) today.

Splice Machine provides built-in system procedures that make it easy to back up and restore your entire database. You can:

- » create full and incremental backups to run immediately
- » schedule daily full or incremental backups
- » restore your database from a backup
- » manage your backups
- » access logs of your backups

The rest of this topic will help you with working with your backups, in these sections:

- » [About Splice Machine Backups](#)
- » [Using the Backup Operations](#)
- » [Backing Up to Cloud Storage](#)

Backup Resource Allocation

Splice Machine backup jobs use a Map Reduce job to copy HFiles; this process may hang up if the resources required for the Map Reduce job are not available from Yarn. See the Backup Resource Allocation section of our *Troubleshooting Guide* for specific information about allocation of resources.

About Splice Machine Backups

Splice Machine supports: both full and incremental backups:

- » A *full backup* backs up all of the files/blocks that constitute your database.
- » An *incremental backup* only stores database files/blocks that have changed since a previous backup.

Because backups can consume a lot of disk space, most customers define a backup strategy that blends their needs for security, recoverability, and space restrictions. Since incremental backups require a lot less space than do full backups, and allow for faster recovery of data, many customers schedule daily, incremental backups.

NOTE: Splice Machine automatically detects when it is the first run of an incremental backup and performs a one-time full backup; subsequent runs will only back up changed files/blocks.

Backup IDs, Backup Jobs, and Backup Tables

Splice Machine uses *backup IDs* to identify a specific full or incremental *backup* that is stored on a file system, and *backup job IDs* to identify each scheduled *backup job*. Information about backups and backup jobs is stored in these system tables:

System Table	Contains Information About
SYS.SYSBACKUP	Each database backup; you can query this table to find the ID of and details about a backup that was run at a specific time.
SYS.SYSBACKUPITEMS	Each item (table) in a backup.
SYS.SYSBACKUPJOBS	Each backup job that has been run for the database.

Temporary Tables and Backups

There's a subtle issue with performing a backup when you're using a temporary table in your session: although the temporary table is (correctly) not backed up, the temporary table's entry in the system tables will be backed up. When the backup is restored, the table entries will be restored, but the temporary table will be missing.

There's a simple workaround:

1. Exit your current session, which will automatically delete the temporary table and its system table entries.
2. Start a new session (reconnect to your database).
3. Start your backup job.

Using the Backup Operations

This section summarizes and provides examples of using the Splice Machine backup operations:

- » [Scheduling a Daily Backup Job](#)
- » [Running an Immediate Backup](#)
- » [Restoring Your Database From a Previous Backup](#)
- » [Reviewing Backup Information](#)

- » [Canceling a Scheduled Backup Job](#)
- » [Canceling a Backup That's In Progress](#)
- » [Deleting a Backup](#)
- » [Deleting Outdated Backups](#)

You must make sure that the directory to which you are backing up or from which data is being restored is accessible to the HBase user who is initiating the restore. Make sure the directory permissions are set correctly on the backup directory.

NOTE: Note that you can store your backups in a cloud-based storage service such as AWS; for more information, see the [Backing Up to Cloud Storage](#) section below.

Scheduling a Daily Backup Job

Use the `SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP` system procedure to schedule a job that performs a daily incremental or full backup of your database:

```
SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP( backupDir, backupType, startHour );
```

backupDir

A VARCHAR value that specifies the path to the directory in which you want the backup stored.

Note that this directory can be cloud-based, as described in the [Backing Up to Cloud Services](#) section below.

backupType

A VARCHAR (30) value that specifies the type of backup that you want performed; one of the following values: `full` or `incremental`. The first run of an incremental backup is always a full backup.

startHour

Specifies the hour (0–23) in GMT at which you want the backup to run each day. A value less than 0 or greater than 23 produces an error and the backup is not scheduled.

Example 1: Set up a full backup to run every day at 3am:

To run a full backup every night at 3am:

```
call SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP( '/home/backup', 'full', 3 );
```

Example 2: Set up an incremental backup to run every day at noon:

To run an incremental backup every day at noon.

```
call SYSCS_UTIL.SYSCS_BACKUP_DATABASE( '/home/backup', 'incremental', 12 );
```

Running an Immediate Backup

Use the `SYSCS_UTIL.SYSCS_BACKUP_DATABASE` system procedure to immediately run a full or incremental backup.

```
SYSICS_UTIL.SYSCS_BACKUP_DATABASE( backupDir, backupType );
```

backupDir

A VARCHAR value that specifies the path to the directory in which you want the backup stored.

Note that this directory can be cloud-based, as described in the [Backing Up to Cloud Services](#) section below.

backupType

A VARCHAR (30) value that specifies the type of backup that you want performed; use one of the following values: **full** or **incremental**.

Example 1: Execute a full backup now

To execute a backup right now:

```
call SYSICS_UTIL.SYSCS_BACKUP_DATABASE( '/home/backup', 'full');
```

Example 2: Execute an incremental backup now:

This call will run an incremental backup right now. Splice Machine checks the **SYSBACKUP** system table to determine if there already is a backup for the system; if not, Splice Machine will perform a full backup, and subsequent backups will be incremental. The backup data is stored in the specified directory.

```
call SYSICS_UTIL.SYSCS_BACKUP_DATABASE( '/home/backup', 'incremental');
```

Restoring Your Database From a Previous Backup

You can restore your database from a previous backup, use the **SYSICS_UTIL.SYSCS_RESTORE_DATABASE** system procedure.

To restore your database from a previous backup, use the **SYSICS_UTIL.SYSCS_RESTORE_DATABASE** system procedure:

```
SYSICS_UTIL.SYSCS_RESTORE_DATABASE(backupDir, backupId);
```

backupDir

A VARCHAR value that specifies the path to the directory in which the backup is stored.

You can find the *backupId* you want to use by querying the **SYSBACKUP** System Table. See the [Reviewing Backup Information](#) section below for more information.

backupId

A BIGINT value that specifies which backup you want to use to restore your database.

There are several important things to know about restoring your database from a previous backup:

- » Restoring a database **wipes out your database** and replaces it with what had been previously backed up.
- » You **cannot use your cluster** while restoring your database.
- » You **must reboot your database** after the restore is complete by first Starting Your Database.

Example: Restore the database from a local, full backup

This example restores your database from the backup stored in the /home/backup directory that has backupId=1273:

```
call SYSCS_UTIL.SYSCS_RESTORE_DATABASE( '/home/backup' , 1273);
```

Reviewing Backups

Splice Machine stores information about your backups and scheduled backup jobs in system tables that you can query, and stores a backup log file in the directory to which a backup is written when it runs.

Backup Job Information

Information about your scheduled backup jobs is stored in the **SYSBACKUPJOBS** system table:

splice> select * from SYS.SYSBACKUPJOBS;				
JOB_ID	FILESYSTEM	TYPE	HOUR_OF_DAY	BEGIN_TIMESTAMP
22275	/data/backup/0101	FULL	22	2015-04-03 18:43:42.6
31				

You can query this table to find a job ID, if you need to cancel a scheduled backup.

Backup Information

Information about each backup of your database is stored in the **SYSBACKUP** system table, including the ID assigned to the backup and its location. You can query this table to find the ID of a specific backup, which you need if you want to restore your database from it, or to delete it:

```
splice> select * from SYS.SYSBACKUP;
BACKUP_ID |BEGIN_TIMESTAMP |END_TIMESTAMP | STATUS |FILESYSTE
M          |SCOPE |INCR&|INCREMENTAL_PARENT_&|BACKUP_ITEM
-----
22275    |2015-04-03 18:40:56.877 |2015-04-03 18:43:42.631 |S      |/data/backup/01
01 |D   |false|-1           |15
21428    |2015-04-03 18:30:55.964 |2015-04-03 18:33:49.494 |S      |/data/backup/01
01 |D   |false|-1           |15
20793    |2015-04-03 18:23:53.574 |2015-04-03 18:27:07.07  |S      |/data/backup/01
01 |D   |false|-1           |87
```

Backup Log Files

When you run a backup, a log file is created or updated in the directory in which the backup is stored. This log file is named `backupStatus.log`, and is stored in plain text, human-readable format. Here's a sample snippet from a log file:

```
Expected time for backup ~12 hours, expected finish at 15:30 on April 8, 2015
5 objects of 833 objects backed up..
6 objects of 833 objects backed up

Finished with Success. Total time taken for backup was 11 hours 32 minutes.
```

Canceling a Scheduled Backup

You can cancel a daily backup with the `SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP` system procedure:

```
SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP( jobId );
```

jobId

A BIGINT value that specifies which scheduled backup job you want to cancel.

You can find the *jobId* you want to cancel by querying the `SYSBACKUPJOBS` system table.

NOTE: Once you cancel a daily backup, it will no longer be scheduled to run.

Example: Cancel a daily backup

This example cancels the backup that has `jobId=1273`:

```
call SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP(1273);
```

Canceling a Backup That's In Progress

You can call the `SYSCS_UTIL.SYSCS_CANCEL_BACKUP` system procedure to cancel a backup that is currently running:

```
SYSCS_UTIL.SYSCS_CANCEL_BACKUP( );
```

Example: Cancel a running backup

This example cancels the Splice Machine backup job that is currently running.

```
call SYSCS_UTIL.SYSCS_CANCEL_BACKUP();
```

Deleting a Backup

Use the [SYSCS_UTIL.SYSCS_DELETE_BACKUP](#) system procedure to delete a single backup:

```
SYSCS_UTIL.SYSCS_DELETE_BACKUP( backupId );
```

backupId

A BIGINT value that specifies which backup you want to delete.

You can find the *backupId* you want to delete by querying the [SYSBACKUP](#) system table,

Example: Delete a backup

This example deletes the backup that has *backupId*=1273:

```
call SYSCS_UTIL.SYSCS_DELETE_BACKUP(1273);
```

Deleting Outdated Backups

Use the [SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS](#) system procedure to delete all backups that are older than a certain number of days.

```
SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS( backupWindow );
```

backupWindow

An INT value that specifies the number of days of backups that you want retained. Any backups created more than *backupWindow* days ago are deleted.

Example: Delete all backups more than a week old

This example deletes all backups that are more than a week old:

```
call SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS(7);
```

Backing Up to Cloud Storage - AWS

You can specify cloud-based directories as destinations for your backups. This section describes how to set up credentials to allow Splice Machine to create and manage backups on AWS.

You need to enable backups by storing your AWS Access Key ID and Secret Access Key values in your cluster's HDFS core-site.xml file: how you set up your credentials depends on the Hadoop platform you are using; see the section below for your platform:

IMPORTANT: You must have access to the S3 bucket to which you are backing up your database. The instructions below give general guidelines; however, S3 access differs in every deployment. For more information, see these sites:

- » <http://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html>
- » <https://wiki.apache.org/hadoop/AmazonS3>

- » [Enabling backups on CDH](#)
- » [Enabling backups on HDP](#)
- » [Enabling backups on MapR](#)

Enabling Splice Machine Backups on CDH

You can use Cloudera Manager to configure properties to enable Splice Machine backups; follow these steps:

1. Navigate to the Cloudera Manager home screen.
2. Stop both HBase and HDFS:
 - » Click the HBase Actions drop-down arrow associated with (to the right of) HBase in the cluster summary section of the home screen, and then click Stop to stop HBase.
 - » Click the HDFS Actions drop-down arrow associated with (to the right of) and then click Stop to stop HDFS.
3. Click **HDFS** in the Cloudera Manager home screen, then click the **Configuration** tab, and in category, click **Advanced**. Then set these property values in the **Cluster-wide Advanced Configuration Snippet (Safety Valve)** for **core-site.xml** field:


```
fs.s3.awsAccessKeyId      = <Your AWS Access Key>
fs.s3.awsSecretAccessKey  = <Your AWS Access Secret Key>
```

4. Restart both services:

- » Click the HDFS Actions drop-down arrow associated with (to the right of) HDFS in the cluster summary

section of the Cloudera Manager home screen, and then click **Start** to restart HDFS.

- » Navigate to the *HBase Status* tab in Cloudera Manager. Then, using the Actions drop-down in the upper-right corner, click **Start** to create a start HBase.

Enabling Splice Machine Backups on HDP

You can use the Ambari dashboard to configure these properties. Follow these steps:

1. **Navigate to the HDFS Configs screen.**
2. **Select the Services tab at the top of the Ambari dashboard screen, then stop both HBase and HDFS:**
 - » Click HBase in the left pane of the screen, then click Service Actions->Stop in the upper-right portion of the Ambari screen.
 - » Click HDFS in the left pane of the screen, the click Service Actions->Stop.
3. **Select Custom core-site and add these properties:**

```
fs.s3.awsAccessKeyId      = <Your AWS Access Key>
fs.s3.awsSecretAccessKey = <Your AWS Secret Access Key>
```
4. **Restart both services:**
 - » Click HDFS in the left pane of the screen, the click Service Actions->Restart All.
 - » Click HBase in the left pane of the screen, the click Service Actions->Restart All.

Enabling Splice Machine Backups on MapR

To enable Amazon S3 access on a MapR cluster, you must stop services, change the configuration files on each node, and then restart services. Follow these steps:

1. **Stop all MapR services by stopping the warden service on each host:**

```
sudo service mapr-warden stop
```
2. **You need to edit two files on each MapR-FS fileserver and HBase RegionServer in your cluster to allow hosts access to Amazon S3. You need to provide the fs.s3 access key ID and secret in each of these files:**

- » /opt/mapr/hadoop/hadoop-2.x.x/etc/hadoop/core-site.xml for Hadoop/MapReduce/YARN 2.x site configuration
- » /opt/mapr/hadoop/hadoop-0.x.x/conf/core-site.xml for Hadoop/MapReduce 0.x/1.x site configuration

If both *MapReduce v1* and *YARN/MapReduce 2* are installed on the MapR compute hosts, the newer *hadoop-2.x.x* version of the file will be canonical, and the older *hadoop-0.x.x* file symbolically linked to it. You can check this using the following ls and file commands:

```
$ ls -al /opt/mapr/hadoop/hadoop-0*/conf/core-site.xml /opt/mapr/hadoop/hadoop-2*/etc/hadoop/core-site.xml
lrwxrwxrwx 1 mapr root 54 Apr 24 11:01 /opt/mapr/hadoop/hadoop-0.20.2/conf/core-site.xml -> /opt/mapr/hadoop/hadoop-2.5.1/etc/hadoop/core-site.xml
-rw-r--r-- 1 mapr root 775 Apr 24 12:50 /opt/mapr/hadoop/hadoop-2.5.1/etc/hadoop/core-site.xml
```

```
$ file /opt/mapr/hadoop/hadoop-0*/conf/core-site.xml /opt/mapr/hadoop/hadoop-2*/etc/hadoop/core-site.xml
/opt/mapr/hadoop/hadoop-0.20.2/conf/core-site.xml: symbolic link to
`/opt/mapr/hadoop/hadoop-2.5.1/etc/hadoop/core-site.xml'
/opt/mapr/hadoop/hadoop-2.5.1/etc/hadoop/core-site.xml: XML document text
```

3. Add your access key ID and secret key in each file by adding the following properties between the <configuration> and </configuration> tags:

```
<!-- AWS s3://bucket/... block-based access -->
<property>
<name>fs.s3.awsAccessKeyId</name>
<value>_AWS_ACCESS_KEY_ID_</value>
</property>
<property>
<name>fs.s3.awsSecretAccessKey</name>
<value>_AWS_SECRET_ACCESS_KEY_</value>
</property>
<!-- AWS s3n://bucket/... filesystem-like access -->
<property>
<name>fs.s3n.awsAccessKeyId</name>
<value>_AWS_ACCESS_KEY_ID_</value>
</property>
<property>
<name>fs.s3n.awsSecretAccessKey</name>
<value>_AWS_SECRET_ACCESS_KEY_</value>
</property>
```

4. Use the `hadoop` command to view your configuration changes:

```
$ hadoop conf | grep fs\\\.s3 | grep -i access | sort -u
<property><name>fs.s3.awsAccessKeyId</name><value>_AWS_ACCESS_KEY_ID_</value><source>core-site.xml</source></property>
<property><name>fs.s3.awsSecretAccessKey</name><value>_AWS_SECRET_ACSES_S_KEY_</value><source>core-site.xml</source></property>
<property><name>fs.s3n.awsAccessKeyId</name><value>_AWS_ACCESS_KEY_ID_</value><source>core-site.xml</source></property>
<property><name>fs.s3n.awsSecretAccessKey</name><value>_AWS_SECRET_ACSES_S_KEY_</value><source>core-site.xml</source></property>
```

5. You can also verify that access is correctly configured with the `hadoop` command to list the contents of an existing bucket. For example:

```
sudo -iu mapr hadoop fs -ls s3n://yourbucketname/
```

6. Finally, restart MapR services on each node via MapR's warden::

```
sudo service mapr-warden start
```

See Also

- » [SYSCS_UTIL.SYSCS_BACKUP_DATABASE](#)
- » [SYSCS_UTIL.SYSCS_CANCEL_BACKUP](#)
- » [SYSCS_UTIL.SYSCS_CANCEL_DAILY_BACKUP](#)
- » [SYSCS_UTIL.SYSCS_DELETE_BACKUP](#)
- » [SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS](#)
- » [SYSCS_UTIL.SYSCS_RESTORE_DATABASE](#)
- » [SYSCS_UTIL.SYSCS_SCHEDULE_DAILY_BACKUP](#)
- » [SYSBACKUP system table](#)
- » [SYSBACKUPITEMS system table](#)
- » [SYSBACKUPJOBS system table](#)

Cleaning Your Database

This is an On-Premise-Only topic! [Learn about our products](#)

Cleaning your database essentially wipes out any user-defined tables, indexes, and related items.



This is a destructive process and should only be used by an administrator.

Following the steps in this topic destroys your database data. And if you have non-SpliceMachine data stored in HBase, you must exercise additional caution to not destroy that data. Please follow the steps for your platform carefully.

You need to follow different steps, depending on which version of Splice Machine you are using:

- » [Cleaning Your Splice Machine Database on a Cloudera-Managed Cluster](#)
- » [Cleaning Your Splice Machine Database on a Hortonworks HDP-Managed Cluster](#)
- » [Cleaning Your Splice Machine Database on a MapR-Managed Cluster](#)
- » [Cleaning Your Splice Machine Database on a Standalone installation](#)

Cleaning Your Splice Machine Database on a Cloudera-Managed Cluster

Follow these steps to clean your database if you're using the Cloudera-managed cluster version of Splice Machine:



This is a destructive process and should only be used by an administrator!

1. Shut down HBase and HDFS

Navigate to the Services->All Services screen in Cloudera Manager, and select these actions to stop HBase and HDFS:

```
hbase -> Actions -> Stop
hdfs1 -> Actions -> Stop
zookeeper1 -> Actions -> Stop
```

2. Use the Zookeeper client to clean things:

Restart ZooKeeper in the Services->All Services screen:

```
zookeeper1 -> Actions -> Start
```

Log in to the machine running Zookeeper on your cluster and start up a command-line (terminal) window.

Run the `zookeeper-client` command. At the prompt, run the following commands:

```
rmr /splice
rmr /hbase
quit
```

3. Start HDFS

Navigate to the Services->All Services screen in Cloudera Manager, and restart *HDFS*:

```
hdfs1 -> Actions -> Start
```

4. Clean up HBase

Use the following shell command to delete the existing `/hbase` directory. You can run this command on any Data Node:

```
sudo -su hdfs hadoop fs -rm -r /hbase
```

If you are logged in as root, use this command instead:

```
sudo -u hdfs hadoop fs -rm -r /hbase
```

NOTE: If the machine running Cloudera Manager is not part of the cluster, **do not** run the command on that machine

5. Create a new HBase directory:

Navigate to the *HBase* screen in Cloudera Manager, and create a new `/hbase` directory by selecting:

```
Actions -> Create Root Directory
```

6. Restart HBase

Now restart HBase from the same Home->Services->`hbase1` screen in Cloudera Manager, using this action:

```
Actions -> Start
```

Cleaning Your Splice Machine Database on a Hortonworks HDP-Managed Cluster

Follow these steps to clean (or *flatten*) your database if you're using Splice Machine on an Ambari-managed Hortonworks Cluster:



This is a destructive process and should only be used by an administrator!

1. Shut down HBase and HDFS

Log in to the Ambari Dashboard by pointing your browser to the publicly visible <hostName> for your master node that is hosting Ambari Server:

```
http://<hostName>:8080/
```

Select these actions to stop HBase and HDFS:

```
Services->HBase->Service Actions->Stop  
Services->HDFS->Service Actions->Stop
```

2. Use the Zookeeper client to clean things

Log in to the node running Zookeeper on your cluster and start up a command-line (terminal) window.

Run the `zookeeper-client` command. At the prompt, run the following commands:

```
rmr /splice  
rmr /hbase  
quit
```

3. Restart HDFS

Use the Ambari Dashboard to restart HDFS:

```
Services->HDFS->Service Actions->Start
```

4. Re-create the required directory structure

You need to SSH into a node that is running the HDFS Client and re-create the directory structure that Splice Machine expects by issuing these commands:

Run the `zookeeper-client` command. At the prompt, run the following commands:

```
sudo -su hdfs hadoop fs -rm -r /apps/hbase  
sudo -su hdfs hadoop fs -mkdir /apps/hbase  
sudo -su hdfs hadoop fs -mkdir /apps/hbase/data  
sudo -su hdfs hadoop fs -chown hbase:hdfs /apps/hbase  
sudo -su hdfs hadoop fs -chown hbase:hdfs /apps/hbase/data
```

5. Restart HBase

Use the Ambari Dashboard to restart HBase:

```
Services->HBase->Service Actions->Start
```

Cleaning Your Splice Machine Database on a MapR-Managed Cluster

Follow the steps below to clean (flatten) your database on your MapR cluster. You must be logged in as the cluster administrator (typically `clusteradmin` or `ec2-user`) to run each step. Unless otherwise specified, run each of these steps **on your cluster control node**; some steps, as indicated, must be run on each node in your cluster.



This is a destructive process and should only be used by an administrator!

1. Stop the HBase RegionServers and Master:

Use the following command **on your control node** to stop HBase on your cluster:

```
~/splice-installer-mapr4.0/stop-hbase.sh
```

2. Remove old data from HDFS:

Ignore any error messages you may see when you run this command:

```
sudo -iu mapr hadoop fs -rm -r -f 'maprfs:///hbase/*'
```

3. Stop MapR warden services:

Run the following command **on each node** in your cluster:

```
sudo service mapr-warden stop
```

4. Launch the ZooKeeper command line shell:

Note that the exact path may vary with different MapR versions

```
/opt/mapr/zookeeper/zookeeper-3.4.5/bin/zkCli.sh
```

5. Connect to the local ZooKeeper instance:

When the ZooKeeper command shell prompts you, enter the **connect** command shown here:

```
Connecting to localhost:2181
Welcome to ZooKeeper!
JLine support is enabled
[zk: localhost:2181(CONNECTING) 0] connect localhost:5181
```

6. Complete the connection:

Press Enter again to display the connected prompt

```
[zk: localhost:5181(CONNECTED) 1]
```

7. Clear old ZooKeeper data:

Enter the following commands to clear ZooKeeper data and then exit the command shell:

```
rnr /splice
rnr /hbase
quit
```

8. Restart MapR warden services on all nodes:

Run the following command **on each node** in your cluster:

```
sudo service mapr-warden start
```

Once you do so, your cluster will re-create the Splice Machine schema, and the command line interface will once again be available after a minute or so.

9. Restart HBase

Run this command to restart hbase:

```
~/splice-installer-mapr4.0/start-hbase.sh
```

Cleaning Your Database in the Standalone Version

Follow these steps to clean your database if you're using the Standalone version of Splice Machine:

1. Make sure that you have quit the `splice>` command line interpreter:

```
splice> quit;
```

2. Change directory to your install directory:

```
cd splicemachine
```

3. Run the following scripts:

```
$ ./bin/stop-splice.sh
$ ./bin/clean.sh
$ ./bin/start-splice.sh
```

Shutting Down Your Database

This is an On-Premise-Only topic! [Learn about our products](#)

This topic describes how to shut down your Splice Machine database. You need to follow different steps, depending on which version of Splice Machine you are using:

- » [Shutting Down Your Splice Machine Database on a Cloudera-Managed Cluster](#)
- » [Shutting Down Your Splice Machine Database on a Hortonworks HDP-Managed Cluster](#)
- » [Shutting Down Your Splice Machine Database on a MapR-Managed Cluster](#)
- » [Shutting Down Your Splice Machine Database on a Standalone installation](#)

Shutting Down Your Splice Machine Database on a Cloudera-Managed Cluster

Use the Cloudera Manager to either shut down HBase or to shut down the entire cluster, whichever is appropriate for your situation.

1. Navigate to the **Services->All Services** screen in *Cloudera Manager*, and select this action to stop **HBase**:

```
hbase -> Actions -> Stop
```

2. If you also want to shut down the entire cluster, select this action in the same screen to stop **HDFS**:

```
hdfs1 -> Actions -> Stop
```

Shutting Down Your Splice Machine Database on a Hortonworks HDP-Managed Cluster

Use the Ambari dashboard to shut down Splice Machine:

1. Log in to the Ambari Dashboard by pointing your browser to the publicly visible <**hostName**> for your master node that is hosting Ambari Server:

```
http://<hostName>:8080/
```

2. Shut down cluster services by selecting:

```
Action -> Stop All
```

Shutting Down Your Splice Machine Database on a MapR-Managed Cluster

To shut down Splice Machine, use the MapR Command System (MCS) to stop HBase.

1. Navigate to the node that is running the `mapr-webserver` service.
2. Log into `https://<hostIPAddr>:8443`, substituting the correct `<hostIPAddr>` value.
3. Stop `hbase`.

Shutting Down Your Database in the Standalone Version

Follow these steps to shut down your database if you're using the Standalone version of Splice Machine:

1. Make sure that you have quit the `splice>` command line interpreter:

```
splice> quit;
```

2. Change directory to your install directory:

```
cd splicemachine
```

3. Run the following scripts:

```
$ ./bin/stop-splice.sh
```

This stops the Splice Machine database and the Splice Machine Administrative Console.

Derby Property Access

This is an On-Premise-Only topic! [Learn about our products](#)

This topic describes how to enable and disable Derby features by setting Derby properties, which are divided into categories, as shown in the following table.

NOTE: When a property value is used, the property is searched for in the order shown in the table; in other words, the property is searched for in the JVM properties; if not found there, it is searched for in the Service properties, then in the Database properties, and finally in the App properties.

Property Type	Description
JVM (System)	<p>These properties can be as command line arguments to JVM:</p> <ul style="list-style-type: none"> » For Maven, include the command line arguments as an <code><argument></code> element in the <code>pom.xml</code> file. For example: <pre><argument>-Dderby.language.logStatementText=true</argument></pre> <ul style="list-style-type: none"> » For shell scripts, you may need to manually add the argument to the java executable command line. For example: <pre>java -Dderby.language.logStatementText=true ...</pre> <p>System properties can also be set manually using the <code>System.setProperty(key, value)</code> function.</p>

Property Type	Description
Service	<p>Service properties are a special type of database property that is required to boot the database; as such, these properties cannot be stored in the database. Instead, they are stored outside the database, as follows:</p> <ul style="list-style-type: none"> » Derby stores service properties in the <code>service.properties</code> file » Splice Machine stores service properties in a Zookeeper element. <p>You can temporarily change service properties with the <code>SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY</code> built-in system procedure. These changes will be lost when the server is next restarted.</p> <p>To make a permanent change in a Derby service property, modify the value in the <code>service.properties</code> file and then restart the server to apply the changes.</p> <p>To make a permanent change in a Splice Machine service property, modify the value in ZooKeeper and then restart the server to apply the changes.</p>
Database	<p>Splice Machine database properties are saved in a hidden HBASE table with CONGLOMERATEID=16.</p>
App (Derby/Splice)	<p>App properties for both Derby and Splice Machine are saved to the <code>derby.properties</code> file in the Derby/Splice home directory.</p> <p>App properties can also be saved to these HBASE XML configuration files:</p> <ul style="list-style-type: none"> » <code>hbase-default.xml</code> » <code>hbase-site.xml</code> » <code>splice-site.xml</code> <p>Note that the XML files must reside in the <code>CLASSPATH</code>.</p>

Site File Example

The following is an example of a `splice-site.xml` file:

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>splice.debug.logStatementContext</name>
    <value>true</value>
    <description>Property to enable logging of all statements.</description>
  </property>
  <property>
    <name>splice.debug.david</name>
    <value>true</value>
    <description>Property to test something.</description>
  </property>
</configuration>

```

Displaying Derby/Splice Properties

You can display the Derby/Splice properties with the `SYSCS_UTIL.SYSCS_GET_ALL_PROPERTIES` system procedure, which displays information like this:

```
splice> call SYSCS_UTIL.SYSCS_GET_ALL_PROPERTIES();
```

KEY	VALUE	TYPE
derby.authentication.builtin.algorithm	SHA-256	DATABASE
derby.user.BROWSE	Browse	APP
derby.engineType	2	SERVICE
derby.david.foo	WINTERS	APP
derby.connection.requireAuthentication	false	JVM
derby.locks.escalationThreshold	500	SERVICE
derby.database.defaultConnectionMode	fullAccess	SERVICE
derby.database.propertiesOnly	false	SERVICE
derby.database.collation	UCS_BASIC	DATABASE
derby.language.logStatementText	false	JVM
derby.storage.propertiesId	16	SERVICE
derby.language.logQueryPlan	true	JVM
splice.updateSystemProcs	false	JVM
derby.storage.rowLocking	false	SERVICE

See Also

- » [SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY](#)
- » The *Derby Properties Guide* on the [Apache Derby documentation site](#).

Enabling Enterprise Features in Splice Machine

This is an On-Premise-Only topic! [Learn about our products](#)

There are two mechanisms for enabling enterprise features in Splice Machine:

- » To access enterprise features such as Backup and Restore, you can simply call the `SYSCS_UTIL.SYSCS_ENABLE_ENTERPRISE` built-in system procedure with the license key you obtain from Splice Machine, as described in the next section.
- » To unlock Splice Machine Enterprise features that require configuration changes in your HBase settings, such as Kerberos and LDAP, you need to add one or more properties to your configuration file, as described in [Using Configuration Properties to Upgrade to the Enterprise](#), below.

Using `SYSCS_UTIL.SYSCS_ENABLE_ENTERPRISE` to Upgrade

If you want to use Enterprise features, such as Backup and Restore, that do not need system properties updated, you can call the `SYCS_UTIL.SYSCS_ENABLE_ENTERPRISE` system procedure to unlock those features. You only need to do this once:

1. Obtain your Splice Machine Enterprise Edition license key.
2. Enter this command on the `splice>` command line:

```
splice> CALL SYSCS_UTIL.SYSCS_ENABLE_ENTERPRISE('<yourLicenseKey>'); Statement executed.
```

If you enter an invalid license key, you'll see an error message:

```
splice> CALL SYSCS_UTIL.SYSCS_ENABLE_ENTERPRISE ('<bogus-license>');
Error
-----
ERROR XSRSE: Unable to enable the enterprise Manager. Enterprise services are disabled. Contact your Splice Machine representative to enable.
```

Using Configuration Properties to Upgrade to the Enterprise

If your site uses Kerberos or LDAP, you need to update to the Enterprise version of Splice Machine by modifying your cluster's HBase configuration, and then restart Splice Machine. Follow these steps:

1. Obtain your Splice Machine Enterprise Edition license key.
2. Edit the `hbase-site.xml` configuration file, adding this property:

```
<property>    <name>splicemachine.enterprise.key</name>    <value><your-Spl  
ice-Machine-license-key></value></property>
```

3. If you're using or switching from another authentication mechanism to LDAP, also add the LDAP properties to your `hbase-site.xml` file, as described in the [Splice Machine Authentication and Authorization](#) topic.
4. If you're using Kerberos, add this to your HBase Master Java Configuration Options:

```
-Dsplice.spark.hadoop.fs.hdfs.impl.disable.cache=true
```

5. Restart Splice Machine, by first Starting Your Database.

Splice Machine Troubleshooting and Best Practices

This is an On-Premise-Only topic! [Learn about our products](#)

This topic provides troubleshooting guidance for these issues that you may encounter with your Splice Machine database:

- » [Restarting Splice Machine after an HMaster Failure](#)
- » [Updating Stored Query Plans after a Splice Machine Update](#)
- » [Increasing Parallelism for Spark Shuffles](#)
- » [Force Compaction to Run Locally](#)
- » [Kerberos Configuration Option](#)
- » [Resource Allocation for Backup Jobs](#)
- » [Bulk Import of Very Large Datasets with Spark 2.2](#)

Restarting Splice Machine After HMaster Failure

If you run Splice Machine without redundant HMasters, and you lose your HMaster, follow these steps to restart Splice Machine:

1. Restart the HMaster node
2. Restart every HRegion Server node

Updating Stored Query Plans after a Splice Machine Update

When you install a new version of your Splice Machine software, you need to make these two calls:

```
CALL SYSCS_UTIL.SYSCS_UPDATE_METADATA_STORED_STATEMENTS();
CALL SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE();
```

These calls will update the stored metadata query plans and purge the statement cache, which is required because the query plan APIs have changed. This is true for both minor (patch) releases and major new releases.

Increasing Parallelism for Spark Shuffles

You can adjust the minimum parallelism for Spark shuffles by adjusting the value of the `splice.olap.shuffle.partitions` configuration option.

This option is similar to the `spark.sql.shuffle.partitions` option, which configures the number of partitions to use when shuffling data for joins or aggregations; however, the `spark.sql.shuffle.partitions` option is set to allow a lower number of partitions than is optimal for certain operations.

Specifically, increasing the number of shuffle partitions with the `splice.olap.shuffle.partitions` option is useful when performing operations on small tables that generate large, intermediate datasets; additional, but smaller sized partitions allows us to operate with better parallelism.

The default value of `splice.olap.shuffle.partitions` is 200.

Force Compaction to Run on Local Region Server

Splice Machine attempts to run database compaction jobs on an executor that is co-located with the serving Region Server; if it cannot find a local executor after a period of time, Splice Machine uses whatever executor Spark executor it can get; to force use of a local executor, you can adjust the `splice.spark.dynamicAllocation.minExecutors` configuration option.

To do so:

- » Set the value of `splice.spark.dynamicAllocation.minExecutors` to the number of Region Servers in your cluster
- » Set the value of `splice.spark.dynamicAllocation.maxExecutors` to equal to or greater than that number.
Adjust these setting in the Java Config Options section of your HBase Master configuration.

The default option settings are:

```
-Dsplice.spark.dynamicAllocation.minExecutors=0
-Dsplice.spark.dynamicAllocation.maxExecutors=12
```

For a cluster with 20 Region Servers, you would set these to:

```
-Dsplice.spark.dynamicAllocation.minExecutors=20
-Dsplice.spark.dynamicAllocation.maxExecutors=20
```

Kerberos Configuration Option

If you're using Kerberos, you need to add this option to your HBase Master Java Configuration Options:

```
-Dsplice.spark.hadoop.fs.hdfs.impl.disable.cache=true
```

Resource Management for Backup Jobs

Splice Machine backup jobs use a Map Reduce job to copy HFiles; this process may hang up if the resources required for the Map Reduce job are not available from Yarn. To make sure the resources are available, follow these three configuration steps:

1. [Configure minimum executors for Splice Spark](#)
2. [Verify that adequate vcores are available for Map Reduce tasks](#)
3. [Verify that adequate memory is available for Map Reduce tasks](#)

Configure the minimum number of executors allocated to Splice Spark

You need to make sure that both of the following configuration settings relationships hold true.

```
(splice.spark.dynamicAllocation.minExecutors + 1) < (yarn.nodemanager.resource.cpu-vcores * number_of_nodes)
```

```
(splice.spark.dynamicAllocation.minExecutors * (splice.spark.yarn.executor.memoryOverhead+splice.spark.executor.memory) + splice.spark.yarn.am.memory) < (yarn.nodemanager.resource.memory-mb * number_of_nodes)
```

The actual `minExecutors` allocated to Splice Spark may be less than specified in `splice.spark.dynamicAllocation.minExecutors` because of memory constraints in the container. Once Splice Spark is launched, Yarn will allocate the actual `minExecutor` value and memory to Splice Spark. You need to verify that enough vcores and memory remain available for Map Reduce tasks.

Verify that adequate vcores are available

The Map Reduce application master requires the following number of vcores:

```
yarn.app.mapreduce.am.resource.cpu-vcores * splice.backup.parallelsim
```

There must be at least this many additional vcores available to execute Map Reduce tasks:

```
max{mapreduce.map.cpu.vcores, mapreduce.reduce.cpu.vcores}
```

Thus, the total number of vcores that must be available for Map Reduce jobs is:

```
yarn.app.mapreduce.am.resource.cpu-vcores * splice.backup.parallelsim + max{mapreduce.map.cpu.vcores, mapreduce.reduce.cpu.vcores}
```

Verify that adequate memory is available

The Map Reduce application master requires this much memory:

```
yarn.scheduler.minimum-allocation-mb * splice.backup.parallelsim
```

There must be at least this much memory available to execute Map Reduce tasks:

```
yarn.scheduler.minimum-allocation-mb
```

Thus, the total number of memory that must be available for Map Reduce jobs is:

```
yarn.scheduler.minimum-allocation-mb * (splice.backup.parallelsim+1)
```

Bulk Import of Very Large Datasets with Spark 2.2

When using Splice Machine with Spark 2.2 with Cloudera, bulk import of very large datasets can fail due to direct memory usage. Use the following settings to resolve this issue:

Update Shuffle-to-Mem Setting

Modify the following setting in the Cloudera Manager's *Java Configuration Options for HBase Master*:

```
-Dsplice.spark.reducer.maxReqSizeShuffleToMem=134217728
```

Update the YARN User Classpath

Modify the following settings in the Cloudera Manager's *YARN (MR2 Included) Service Environment Advanced Configuration Snippet (Safety Valve)*:

```
YARN_USER_CLASSPATH=/opt/cloudera/parcels/SPARK2/lib/spark2/yarn/spark-2.2.0.clouder  
a1-yarn-shuffle.jar:/opt/cloudera/parcels/SPARK2/lib/spark2/jars/scala-library-2.1  
1.8.jar  
YARN_USER_CLASSPATH_FIRST=true
```

About our Configuration Options

This is an On-Premise-Only topic! [Learn about our products](#)

The following table provides summary information about (some of) the configuration options used by Splice Machine.

Property	Typical value	Description
splice.authentication	NATIVE	This is documented in our Configuring Authentication topic.
splice.authentication.native.algorithm	SHA-512	This is documented in our Configuring Authentication topic.
splice.client.write.maxDependentWrites	60000	A form of write throttling that controls the maximum number of concurrent dependent writes from a single process. Dependent writes are writes to a table with indexes and generate more independent writes (writes to the indexes themselves). They are segregated so we can guarantee progress by reserving some IPC threads for independent writes.
splice.client.write.maxIndependentWrites	60000	A form of write throttling that controls the maximum number of concurrent independent writes from a single process.
splice.compression	snappy	The type of compression to use when compressing Splice Tables. This is set the same HBase sets table compression, and has the same codecs available to it (GZIP,Snappy, or LZO depending on what is installed).
splice.debug.logStatementContext	false	Whether to log all statements. Note that this is costly in OLTP workloads.
splice.marshal.kryoPoolSize	1100	The maximum number of Kryo objects to pool for reuse. This setting should only be adjusted if there are an extremely large number of operations allowed on the system.
splice.olap_server.clientWaitTime	900000	The number of milliseconds the OLAP client should wait for a result.

Property	Typical value	Description
splice.root.path	/splice	Zookeeper root node for Splice Machine. All Zookeeper nodes used by Splice Machine will hang from this root node.
splice.splitBlockSize	67108864	Default size for Spark partitions when reading data from HBase. We sub-split each HBase region into several partitions targeting that size, but it rarely matches exactly. This is because it depends on a number of factors, including the number of storefiles in use by a given HBase region.
splice.timestamp_server.clientWaitTime	120000	The number of milliseconds the timestamp client should wait for a result.
splice.txn.completedTxns.concurrency	128	Concurrency level for the completed transactions cache. Specifies the estimated number of concurrently updating threads.
splice.txn.concurrencyLevel	4096	Expected concurrent updates to a transaction region. Increasing it increases memory consumption, decreasing it decreases concurrency on transaction operations. A reasonable default is the number of ipc threads configured for this system.

Release Notes for Splice Machine

Welcome to the 2.7 Release of Splice Machine, originally released March, 2018. The release notes for our database are presented in these topics:

- » New Features and Changes
- » Improvements
- » Issues Fixed
- » Limitations and Workarounds

Each of the above topics contains the release notes for the initial Release, along with links to release notes for incremental (patch) versions that have been released since then.



Most of the features incorporated into the release 2.7 of Splice Machine have also been backported into version 2.5.1. The documentation and release notes for v2.5.1 are found here: <https://doc.splicemachine.com/2.5.1>

New Features and Changes in Release 2.7 of the Splice Machine Database

The Splice Machine database is used in both our Database-Service and On-Premise Database products.

This page describes the major new features added to the Splice Machine database since the 2.5 GA Release of Splice Machine. With the exception of a few features note in the [New Features Only Available in v2.7 section](#) below, all of these features have added to both the 2.7 and 2.5.1 Releases of the Splice Machine database.



Splice Machine Release 2.6 was an interim release in September, 2017, which coincided with the initial release of our Database-as-a-Service product. All changes in v2.6 have also been incorporated into both the 2.7 and 2.5.1 Releases of the Splice Machine database.

New Features Only Available in Splice Machine Release 2.7

These features have been added to Release 2.7 of Splice Machine, and have not been backported to Release 2.5.1:

JIRA-ID	Description
Internal	Add support for AVRO format for external tables.
Internal	Query Logging
Internal	Add MacOS version of our ODBC driver. <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> This is a provisional feature that has not yet been as thoroughly tested as it will be. </div>
Internal	JDBC Driver Improvements <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> This is a provisional feature that has not yet been as thoroughly tested as it will be. </div>

New Features in Available in Splice Machine Releases 2.7 and 2.5.1

Each section in this topic includes a list of major new in each interim Splice Machine release since the 2.5 GA Release (2.5.0.1707) on March 1, 2017.

- » [Patch Release 2.5.0.1805](#)
- » [Patch Release 2.5.0.1804](#)
- » [Patch Release 2.5.0.1803](#)
- » [Patch Release 2.5.0.1802](#)
- » [Patch Release 2.5.0.1749](#)
- » [Patch Release 2.5.0.1748](#)
- » [Patch Release 2.5.0.1747](#)
- » [Patch Release 2.5.0.1745](#)
- » [Patch Release 2.5.0.1735](#)
- » [Patch Release 2.5.0.1729](#)

2.5.0.1805 Patch Release 5-Feb-18

JIRA-ID	Description
Internal	Improve export performance
Internal	Fix illegal merge join
Internal	Prevent incremental backup from losing changes

2.5.0.1804 Patch Release 28-Jan-18

JIRA-ID	Description
Internal	Add hint useDefaultRowCount and defaultSelectivityFactor
Internal	Fix upsert index
Internal	Vacuum disabled tables
Internal	Cache more database properties
Internal	Fix database restore from s3
Internal	Retry write workload on host unreachable exceptions

2.5.0.1803 Patch Release 21-Jan-18

JIRA-ID	Description
Internal	Add system procedure to enable/disable all column statistics
Internal	Close FutureIterator to avoid race condition

2.5.0.1802 Patch Release 14-Jan-18

JIRA-ID	Description
Internal	Clean up incremental changes from offline regions that were offline before full backup
Internal	Add ignore txn cache
Internal	Handle CannotCommit exception correctly
Internal	Reduce lengthy lineage of transformation for MultiProbeTableScan operation
Internal	Add JDBC timeout support
Internal	Get keytab file name correctly

2.5.0.1749 Patch Release 26-Dec-17

JIRA-ID	Description
Internal	Exclude metrics jar for cdh platforms
Internal	Add missing getEncodedName() method

2.5.0.1748 Patch Release 18-Dec-17

JIRA-ID	Description
Internal	Fix inconsistent selectivity with predicate order change
Internal	Delete data by region without scanning

JIRA-ID	Description
Internal	Adjust selectivity estimation by excluding skewed default values
Internal	Propagate SORT elimination to outer joins
SPLICE-1920	Tooling to fix data dictionary corruption

2.5.0.1747 Patch Release 08-Dec-17

JIRA-ID	Description
Internal	Terminate backup timely if it has been cancelled
Internal	Prepend schemaname to columns in PK conditions in update statement generated by MERGE_DATA_FROM_FILE
Internal	Fix column misalignment for join update through mergesort join
Internal	Bound the outer join row count by the left outer table's row count
Internal	Parameter sanity check for region operations
Internal	Populate default value for column of DATE type in defaultRow with the right type (regression fix for SPLICE-1700)
SPLICE-1700	Avoid updating existing physical rows when adding not-null
SPLICE-1802	Create index by HFile loading(2.5)

2.6.1.1745 Patch Release 08-Dec-17



2.6 was an interim release that has been folded in to 2.5

JIRA-ID	Description
Internal	Add region name and id to compaction job description
Internal	Add more logging

JIRA-ID	Description
Internal	Enhance MultiProbeIndexScan to apply for cases where inlist is not the leading index/PK column
Internal	Add logs around task failure in Spark
Internal	Make table level select privileges to column level
Internal	Add back synchronous backup/restore
Internal	Add check for actionAllowed
Internal	Exclude flatten and restart scripts
Internal	Move the bulkimport/bulkdelete test in DefaultIndexIT to a separate file under hbase_sql so it won't run for mem platform
Internal	Add parameter to ignore missing transactions from SPLICE_TXN
Internal	Keep alive backup
SPLICE-1802	Create index by HFile loading
Internal	Synchronize access to shared ArrayList
Internal	Fix automatic DROP of temp tables
Internal	Enhance subquery processing logic to convert eligible where subqueries
Internal	Allow first connection on Kerberized cluster
Internal	Large variance in simple select query execution time
SPLICE-1700	Avoid updating existing physical rows when adding not-null column with default value
Internal	Allow distributed execution for splice client
Internal	Avoid index creation timeout
Internal	Flatten correlated Scalar Subquery in Select clause
Internal	Null checking for region operations
Internal	Avoid sort if some of the index columns are bound with constant
Internal	Set rowlocation for index lookup

Splice Patch Release 2.5.0.1735

JIRA ID	Description
SPLICE-31	Our JDBC driver now supports Kerberos authentication.
SPLICE-662	Our JDBC client now distributes query requests across multiple nodes without losing transactions, which provides a more even distribution of work across the cluster, resulting in better performance.
SPLICE-1320	Adds <code>ARRAY</code> data type
SPLICE-1372	We've added two new system procedures for working with database operations that are currently running on a server: <ul style="list-style-type: none"> » The <code>SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS</code> system procedure displays a list of the operations running on the server to which you are currently connected. » You can use this procedure to find the UUID for an operation, which you can then use for purposes such as terminating an operation with the <code>SYSCS_UTIL.SYSCS_KILL_OPERATION</code> system procedure.
SPLICE-1373	We've made two significant changes in logging: <ul style="list-style-type: none"> » Statement logging is now enabled by default. To modify this, see the Using Logging topic. » Logs are stored, by default, in your region server's log directory. You can modify the location of the logs with a code snippet that is described in the Installation Guide page for your platform (Cloudera, Hortonworks, or MapR).
SPLICE-1482	Improves import performance by using Hfiles.
SPLICE-1512	Allows use of multiple <code>DISTINCT</code> operations in aggregates.
SPLICE-1591	Adds an option to purge deleted rows when performing a major compaction.
SPLICE-1603	Adds a clause to the <code>ANALYZE TABLE</code> command to allow analyzing via sampled statistics, rather than analyzing the entire table.
SPLICE-1607	You can now configure and use SSL/TLS secured JDBC connections to your Splice Machine database. We support both basic encrypted connections and peer-authenticated connections. Please see our Configuring SSL/TLS topic for more information.
SPLICE-1617	Adds support for suppressing <code>NULLs</code> when creating an index to address a situation where most rows in a table contain the default (<code>NULL</code>) value, which wastes storage and causes bad distribution of rows across regions.
SPLICE-1669	Supports bulk delete via HFiles, bypassing the HBase write path.

JIRA ID	Description
SPLICE-1671	Added the ability to take snapshots of tables (and their indexes).
SPLICE-1701	Adds ability to monitor memory usage by HBase JVM through JMX.
SPLICE-1760	The new SYSCS_UTIL.GET_SESSION_INFO system procedure reports information about your current session, including the hostname and session ID for your current session. You can use this information to correlate your Splice Machine query with a Spark job: the same information is displayed in the Job Id (Job Group) in the Spark console.
Internal	We've added a number of built-in system procedures that allow you to work directly with table and index regions; some customers use these to optimize processing of recent data while optimizing storage of older data: <ul style="list-style-type: none"> » SYSCS_UTIL.COMPACT_REGION » SYSCS_UTIL.GET_ENCODED_REGION_NAME » SYSCS_UTIL.GET_REGIONS » SYSCS_UTIL.GET_START_KEY » SYSCS_UTIL.MAJOR_COMPACT_REGION » SYSCS_UTIL.MERGE_REGIONS
Internal	Imports data without overwriting generated or default values contained in non-imported columns.
Internal	Our ODBC Driver now supports BLOB and CLOB objects.

Splice Patch Release 2.5.0.1729 01-Aug-17

JIRA ID	Description
Internal	add a system procedure MERGE_DATA_FROM_FILE to achieve a limited fashion of merge-into
Internal	Implement Kerberos JDBC support (2.5)
SPLICE-1482	HBase Bulk Import
SPLICE-1591	Allow Physical Deletes in a Table
SPLICE-1603	Support sample statistics collection (via Analyze)
SPLICE-1671	Enable snapshot with bulk load procedure

For a full list of JIRA's for the Community/Open Source software, see <https://splice.atlassian.net>

Improvements in Release 2.7 of the Splice Machine Database

The Splice Machine database is used in both our Database-Service and On-Premise Database products.

This page describes all of the significant improvements made in the Splice Machine database since the 2.5 GA Release of Splice Machine. With only one noted exception, all of the improvements listed here have been made in both the 2.7 and 2.5.1 Releases of the Splice Machine database.



Splice Machine release 2.6 was an interim Release in September, 2017, which coincided with the initial Release of our Database-as-a-Service product. All changes in v2.6 have also been incorporated into both the 2.7 and 2.5.1 Releases of the Splice Machine database.

Each section in this topic includes a list of significant improvements in each interim Splice Machine Release since the 2.5 GA Release (2.5.0.1707) on March 1, 2017.

- » [Patch Release 2.5.0.1805](#)
- » [Patch Release 2.5.0.1804](#)
- » [Patch Release 2.5.0.1803](#)
- » [Patch Release 2.5.0.1802](#)
- » [Patch Release 2.5.0.1749](#)
- » [Patch Release 2.5.0.1748](#)
- » [Patch Release 2.5.0.1747](#)
- » [Patch Release 2.5.0.1745](#)
- » [Patch Release 2.5.0.1735](#)
- » [Patch Release 2.5.0.1729](#)

2.5.0.1805 Patch Release 5-Feb-18

JIRA-ID	Description
SPLICE-2022	spark job description for compaction

2.5.0.1804 Patch Release 28-Jan-18

JIRA-ID	Description
No improvements in this patch release.	

2.5.0.1803 Patch Release 21-Jan-18

JIRA-ID	Description
SPLICE-1987	Add fully qualified table name into HBASE TableDisplayName Attribute

2.5.0.1802 Patch Release 14-Jan-18

JIRA-ID	Description
SPLICE-1975	Allow JavaRDD<Row> to be passed for CRUD operations in SplicemachineContext
SPLICE-1991	Add info to Spark UI for Compaction jobs to indicate presence of Reference Files

2.5.0.1749 Patch Release 26-Dec-17

JIRA-ID	Description
SPLICE-1973	Exclude Kafka jars from splice-uber.jar for all platforms
SPLICE-1984	Parallelize MultiProbeTableScan and Union Operations

2.5.0.1748 Patch Release 18-Dec-17

JIRA-ID	Description
No improvements in this patch release.	

2.5.0.1747 Patch Release 08-Dec-17

JIRA-ID	Description
SPLICE-1351	Upgrade Sketching Library from 0.8.1 - 0.8.4
SPLICE-1948	Increase test timeout
SPLICE-1948	Initialize Splice Spark context with user context
SPLICE-1951	Remove protobuf installation instructions

2.6.1.1745 Patch Release 08-Dec-17



2.6 was an interim release that has been folded in to 2.5

JIRA-ID	Description
SPLICE-1302	Add minimum parallelism for Spark shuffles
SPLICE-1951	Remove protobuf installation instructions from
SPLICE-1948	Initialize Splice Spark context with user context
SPLICE-1948	Increase test timeout

Splice Patch Release 2.5.0.1735

JIRA ID	Description
SPLICE-398	Support drop view if exists
SPLICE-949	Built-in function ROUND added
SPLICE-1222	Implement in-memory subtransactions
SPLICE-1351	Upgrade Sketching Library from 0.8.1 - 0.8.4
SPLICE-1372	Control-side query control
SPLICE-1479	Iterator based stats collection
SPLICE-1497	Add flag for inserts to skip conflict detection
SPLICE-1500	Skip WAL for unsafe imports
SPLICE-1513	Create Spark Adapter that supports both 1.6.x and 2.1.0 versions of Spark
SPLICE-1516	Enable compression for WritePipeline
SPLICE-1517	Orc Reader Additions <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;">  This improvement has not been backported into release 2.5.1. </div>
SPLICE-1555	Enable optimizer trace info for costing
SPLICE-1568	Core Spark Adapter Functionality With Maven Build
SPLICE-1619	Update the Spark Adapter to 2.1.1
SPLICE-1681	Introduce query hint "skipStats" after a table identifier to bypass fetching real stats from dictionary tables
SPLICE-1702	Removed LocatedRow construct from the execution tree
SPLICE-1703	Changed size==0 to isEmpty()
SPLICE-1714	Ignore "should not give a splitkey that equates to startkey" exception
SPLICE-1725	External table documentation updated

JIRA ID	Description
SPLICE-1729	Handle 'drop table table_name if exists'
SPLICE-1733	Support type conversion Varchar to INT
SPLICE-1739	Added CREATE SCHEMA IF NOT EXISTS functionality
SPLICE-1752	Support inserting int types to char types
SPLICE-1756	Introduce database property collectIndexStatsOnly to specify the collect stats behavior
SPLICE-1760	Enhancement to provide corresponding Spark JobID when Splice jobs or queries are submitted through Spark
SPLICE-1785	Too many tasks are launched in the last stage of bulk import
SPLICE-1834	Remove EFS FileSystem
SPLICE-1835	Remove MBeanResultSet
SPLICE-1836	Remove SpliceCsvTokenizer
SPLICE-1837	Remove Old Cost Estimate Implementation.
SPLICE-1838	Remove Left Over Aggregate Plumbing
SPLICE-1839	Remove Serial Encoding Package
SPLICE-1840	Remove Dead PhysicalStatsStore
SPLICE-1841	Remove ScanInfo class and Interfaces
SPLICE-1842	Derby Utils Dead Code Cleanup
SPLICE-1845	Tweak Kryo Serde for Missing Elements
SPLICE-1851	Remove concurrent.traffic package
SPLICE-1873	Added documentation for GET_SESSION_INFO
SPLICE-1875	Added documentation for GET_RUNNING_OPERATIONS and KILL_OPERATION
SPLICE-1879	KeyBy Function on Control is a multimap index vs. a map function
SPLICE-1880	Modify ReduceByKey to execute lazily and not use Multimaps.
Internal	Added SplicemachineContext.g

JIRA ID	Description
Internal	Bcast implementation dataset vs <code>rddetConnection()</code> to enable commit/rollback in Scala
Internal	Bcast implementation dataset vs rdd
Internal	Add logging to Vacuum process

Splice Patch Release 2.5.0.1729 01-Aug-17

JIRA ID	Description
SPLICE-398	Support 'drop view if exists' for 2.5
SPLICE-774	Support upgrade from K2 (2.5)
SPLICE-1479	iterator based stats collection (2.5)
SPLICE-1516	Enable compression for WritePipeline
SPLICE-1500	Skip WAL for unsafe imports (2.5)
SPLICE-1555	Enable optimizer trace info for costing
SPLICE-1681	Introduce query hint "skipStats"
SPLICE-1701	JXM mbean server for cache and enginedriver exec service
SPLICE-1729	Support 'drop table t_name if exists' for 2.5
SPLICE-1756	introduce database property collectIndexStatsOnly to specify the collect stats behavior
SPLICE-1769	Improve distributed boot process
Internal	Bcast implementation dataset vs rdd
Internal	prune query blocks based on unsatisfiable conditions
Internal	Add logging to Vacuum process
Internal	Restore breaks transaction semantics(2.5)

For a full list of JIRA's for the Community/Open Source software, see <https://splice.atlassian.net>

Bug Fixes in Release 2.7 of the Splice Machine Database

The Splice Machine database is used in both our Database-Service and On-Premise Database products.

This page describes all of the issues that have been fixed in the Splice Machine database since the 2.5 GA Release of Splice Machine. With only one noted exception, all of the fixes listed here have been applied to both the 2.7 and 2.5.1 Releases of the Splice Machine database.



Splice Machine Release 2.6 was an interim release in September, 2017, which coincided with the initial Release of our Database-as-a-Service product. All changes in v2.6 have also been incorporated into both the 2.7 and 2.5.1 Releases of the Splice Machine database.

Each section in this topic includes a list of issues fixed in each interim Splice Machine Release since the 2.5 GA Release (2.5.0.1707) on March 1, 2017.

- » [Patch Release 2.5.0.1805](#)
- » [Patch Release 2.5.0.1804](#)
- » [Patch Release 2.5.0.1803](#)
- » [Patch Release 2.5.0.1802](#)
- » [Patch Release 2.5.0.1749](#)
- » [Patch Release 2.5.0.1748](#)
- » [Patch Release 2.5.0.1747](#)
- » [Patch Release 2.5.0.1745](#)
- » [Patch Release 2.5.0.1735](#)
- » [Patch Release 2.5.0.1729](#)

2.5.0.1805 Patch Release 5-Feb-18

JIRA-ID	Description
SPLICE-1995	Correct imported rows

2.5.0.1804 Patch Release 28-Jan-18

JIRA-ID	Description
SPLICE-1717	Asynchronous transaction resolution in compactions
SPLICE-2012	HMaster doesn't exit after shutdown

2.5.0.1803 Patch Release 21-Jan-18

JIRA-ID	Description
SPLICE-1900	Incorrect error message while reading data from Empty external table of AVRO file format.  This improvement has not been backported into release 2.5.1.
SPLICE-2004	Fix inconsistency between plan logged in log and that in the explain

2.5.0.1802 Patch Release 14-Jan-18

JIRA-ID	Description
SPLICE-1870	Fix update through index lookup path
SPLICE-1927	Amend pattern string for detecting splice machine ready to accept connections
SPLICE-1998	Modify splice 2.5 log file's time stamp format to ISO8601

2.5.0.1749 Patch Release 26-Dec-17

JIRA-ID	Description
SPLICE-1983	OOM in Spark executors while running TPCH1 repeatedly

2.5.0.1748 Patch Release 18-Dec-17

JIRA-ID	Description
SPLICE-1970	Exclude metrics jars from splice-uber.jar to avoid class loader issues when using Spark Adapter
SPLICE-1978	Add null check to GET_RUNNING_OPERATIONS

2.5.0.1747 Patch Release 08-Dec-17

JIRA-ID	Description
SPLICE-865	Check if enterprise version is activated and if the user try to use column privileges.
SPLICE-1784	Query does not scale on 4000 regions (2.5)
SPLICE-1895	Wait for wrap up before closing remote query client (2.5)
SPLICE-1906	Make coprocessors throw only IOExceptions
SPLICE-1908	Add check for actionAllowed
SPLICE-1921	Fix wrong result with sort merge inclusion join for spark path
SPLICE-1928	Decode region start/end key by fetching actual rowkey
SPLICE-1930	Fixes an issue where maven uses platform installed protobuf
SPLICE-1934	WordUtils.wrap() from commons-lang3 3.5 is broken
SPLICE-1937	Add cdh5.12 platform (2.5)
SPLICE-1961	Missing splice_spark module in pom.xml

2.6.1.1745 Patch Release 08-Dec-17



2.6 was an interim release that has been folded in to 2.5

JIRA-ID	Description
SPLICE-1784	Trying to do cutpoints again
SPLICE-1887	Fix memory leak for merge join
SPLICE-1906	Make coprocessors throw only IOExceptions
SPLICE-1915	Enable query logging by default
SPLICE-1921	Fix wrong result with sort merge inclusion join for spar
SPLICE-1895	Wait for wrap up before closing remote query client
SPLICE-1928	Decode region start/end key by fetching actual rowkey
SPLICE-1934	WordUtils.wrap() from commons-lang3 3.5 is broken

Splice Patch Release 2.5.0.1735

JIRA ID	Issue Description
SPLICE-8	Make CTAS work with case sensitive names
SPLICE-57	Drop backing index and write handler when dropping a unique constraint
SPLICE-77	Order by column in subquery not projected should not be resolved
SPLICE-79	Generate correct insert statement to import char for bit column
SPLICE-612	Fix wrong result in right outer join with expression in join condition
SPLICE-737	GetTableName should return -1 if it cannot be determined
SPLICE-835	Mapping TEXT column creation to CLOB
SPLICE-865	Check if enterprise version is activated and if the user try to use column privileges.

JIRA ID	Issue Description
SPLICE-976	Do not allow odbc and jdbc requests other than splice driver
SPLICE-1023	Add more info in the message for data import error
SPLICE-1062	Retry if region location cannot be found
SPLICE-1098	Prevent nonnull selectivity from being 0
SPLICE-1116	Fixing OrderBy Removal from set operations since they are now hashed based
SPLICE-1122	Deleting a table needs to remove the pin for the table.
SPLICE-1160	Report syntax error instead of throwing NPE for topN in set operation
SPLICE-1208	Fix mergeSortJoin overestimate by 3x
SPLICE-1211	Open and close latency calculated twice for NLJ
SPLICE-1290	Adding Batch Writes to the dictionary
SPLICE-1294	Poor Costing when first part of PK is not =
SPLICE-1320	Fix Assertion Placement and Comment Out test to get build out
SPLICE-1323	Add null check for probevalue
SPLICE-1324	Making Sure SQLTinyInt can SerDe
SPLICE-1329	Memory leak in SpliceObserverInstructions
SPLICE-1349	Serialize and initialize BatchOnceOperation correctly
SPLICE-1353	Export to S3
SPLICE-1357	Fix wrong result for right outer join that is performed through spark engine
SPLICE-1358	CREATE EXTERNAL TABLE can failed with some specific users in hdfs,
SPLICE-1359	SanityManager.DEBUG messages create a lot of noise in derby.log
SPLICE-1360	Adding SQL Array Data Type Basic Serde Functions
SPLICE-1361	Kerberos keytab not picked up by Spark on Splice Machine
SPLICE-1362	Synchronize access to internalConnection's contextManager
SPLICE-1369	Store external table on S3

JIRA ID	Issue Description
SPLICE-1370	INSERT, UPDATE, DELETE error message for pin tables
SPLICE-1374	Bad file in S3
SPLICE-1375	Fix concurrency issues reporting failedRows
SPLICE-1379	The number of threads in the HBase priority executor is inadequately low
SPLICE-1386	Load jar file from S3
SPLICE-1395	Add a generic error message for import failure from S3
SPLICE-1410	Make the compilation of the pattern static
SPLICE-1411	Resolve over clause expr when alias from inner query is used in window fn
SPLICE-1423	Add null check for bad file directory
SPLICE-1424	Removing Unneeded Visitor from FromTable
SPLICE-1425	Fix data type inconsistencies with unary functions and external table based on TEXT data format
SPLICE-1430	Remove pin from dictionary and rely on spark cache to get the status of pins & Fix race condition on OlapNIOLayer
SPLICE-1433	Fix drop pinned table
SPLICE-1438	Fix the explain plan issues
SPLICE-1443	Add logging to debug "can't find subpartitions" exception
SPLICE-1443	Skip cutpoint that create empty partitions
SPLICE-1446	Check schema for ext table only if there's data
SPLICE-1448	Make sure SpliceSpark.getContext/Session isn't misused
SPLICE-1452	Correct cardinality estimation when there is missing partition stats
SPLICE-1453	Fixing Calculating Stats on Array Types
SPLICE-1461	Add null check on exception parsing
SPLICE-1461	Wrap exception parsing against errors
SPLICE-1462	Adding Mesos Scheduling Option to Splice Machine
SPLICE-1463	Sort results in MemStoreKVScanner when needed

JIRA ID	Issue Description
SPLICE-1464	Bypass schema checking for csv file
SPLICE-1469	Set hbase.rowlock.wait.duration to 0 to avoid deadlock
SPLICE-1470	Make sure user transaction rollbacks on Spark failure
SPLICE-1473	Allow user code to load com.splicemachine.db.iapi.error
SPLICE-1478	Fixing Statement Limits
SPLICE-1479	iterator based stats collection
SPLICE-1480	Allow N Tree Logging
SPLICE-1481	Unnecessary Interface Modifier
SPLICE-1489	Make Predicate Pushdown defaulted for ORC
SPLICE-1490	Bringing Derby Style Forward
SPLICE-1491	Remove Array Copy for Key From Insert
SPLICE-1526	Handle CodecPool manually to avoid leaking memory
SPLICE-1531	Fixed ThreadLocal in AbstractTimeDescriptorSerializer
SPLICE-1533	Eliminate duplicates in the IN list
SPLICE-1541	Fix IN-list issues with dynamic bindings and char column
SPLICE-1543	
SPLICE-1550	Recursive Init Calls
SPLICE-1559	bulkImportDirectory is case sensitive
SPLICE-1561	Allowing Clients to turn off cache and lazily execute
SPLICE-1567	Set remotecost for merge join
SPLICE-1578	Upgrade from 2.5 to 2.7
SPLICE-1582	Apply memory limit on consecutive broadcast joins
SPLICE-1584	Fix IndexOutOfBoundsException when not all column stats are collected and we try to access column stats for estimation.

JIRA ID	Issue Description
SPLICE-1586	Prevent NPE when Spark job fails
SPLICE-1589	All transactions are processed by pre-created region 0
SPLICE-1597	Fix issue with cache dictionary when SYSCS_UTIL.SYSCS_UPDATE_SCHEMA_OWNER is called
SPLICE-1601	Fix wrong result for min/max/sum on empty table without groupby
SPLICE-1609	Normalize row source for split_table_or_index procedure
SPLICE-1611	TPC-C workload causes many prepared statement recompilations
SPLICE-1613	Ignore saveSourceCode IT for now
SPLICE-1621	Fix select from partitioned orc table error
SPLICE-1622	Return only latest version for sequences
SPLICE-1624	Load pipeline driver at RS startup
SPLICE-1628	Don't raise exception if path doesn't exist
SPLICE-1628	Parallelize hstore bulkLoad step in Spark
SPLICE-1637	Enable compression for HFile gen in bulk loader
SPLICE-1639	Fix NPE due to Spark static initialization missing
SPLICE-1640	Apply memory limit check for consecutive outer broadcast join and derived tables
SPLICE-1660	Delete Not Using Index Scan due to index columns being required for the scan.
SPLICE-1675	Merge partition stats at the stats collection time
SPLICE-1682	Perform accumulator check before txn resolution
SPLICE-1684	Fix stats collection logic for ArrayIndexOutOfBoundsException in the presence of empty partition and some column stats disabled
SPLICE-1690	Merge statistics on Spark
SPLICE-1696	Add ScanOperation and SpliceBaseOperation to Kryo
SPLICE-1698	StringBuffer to StringBuilder
SPLICE-1699	Removing Unused Imports

JIRA ID	Issue Description
SPLICE-1703	Replace size() == 0 with isEmpty()
SPLICE-1704	Replace double quotes with isEmpty
SPLICE-1707	Fix Tail Recursion Issues
SPLICE-1708	Do not use KeySet where entryset will work
SPLICE-1711	Replace concat with +
SPLICE-1712	Remove Constant Array Creation Style
SPLICE-1737	Fix value outside the range of the data type INTEGER error for analyze table statement.
SPLICE-1744	Removing Dictionary Check
SPLICE-1748	Fixing Role Cache Usage
SPLICE-1749	Fix delete over nestedloop join
SPLICE-1759	HBase Master generates 1.1GB/s of network bandwidth even when cluster is idle
SPLICE-1769	Improve distributed boot process
SPLICE-1773	Unifying the thread pools
SPLICE-1781	Fixing Object Creation on IndexTransformFunction
SPLICE-1784	Fixing Serial Cutpoint Generation
SPLICE-1784	Query does not scale on 4000 regions
SPLICE-1791	Make username's more specific to resolve concurrent conflicts
SPLICE-1792	BroadcastJoinMemoryLimitIT must be executed serially
SPLICE-1795	Fix NullPointerException for update with expression, and uncomment test case in HdfsImport related to this bug
SPLICE-1798	Parallel Queries can fail on SPS Descriptor Update...
SPLICE-1813	Transaction are not popped from transaction stack when releasing savepoints
SPLICE-1824	NullPointerException when collecting stats on ORC table
SPLICE-1850	Couldn't find subpartitions in range exception with external tables
SPLICE-1853	Wrong count(*) result for partitioned external table

JIRA ID	Issue Description
SPLICE-1854	wrong result query orc table with predicate on date column
SPLICE-1858	Join result is wrong for a partitioned external table
SPLICE-1860	Error analyzing table when columns contains zero length data
SPLICE-1865	Boolean operator <> is broken with external tables
SPLICE-1867	SHOW TABLES is broken
SPLICE-1874	Large table scan runs on control with predicate of high selectivity
Internal	Allow more packages to be loaded from user code
Internal	Drop and re-create foreign key write handler after truncating a table
Internal	Name space null check
Internal	Fix table number to allow predicate push down
Internal	Remove check for collecting schema level stats for external table
Internal	Explicitly unset ordering
Internal	Redo nested connection on Spark fix
Internal	Avoid bad file naming collision
Internal	Allow inner table of broadcast join to be any FromTable
Internal	Fix stat collection on external table textfile
Internal	Resubmit to Spark if we consume too many resources in control
Internal	Fix a couple issues that cause backup to hang
Internal	Clean up timeout backup
Internal	Bind select statement only once in insert into select
Internal	Concatenate all iterables at once to avoid stack overflow error
Internal	Prune query blocks based on unsatisfiable conditions
Internal	Fix hash join column ordering
Internal	Throw BR014 for concurrent backup

JIRA ID	Issue Description
Internal	Fix incremental backup hang
Internal	Restore cleanup
Internal	Continue processing tables when one doesn't have a namespace
Internal	Restore a chain of backup(2.5)
Internal	Correct postSplit
Internal	Fixing S3 File System Implementation
Internal	Allowing SpliceClient.isClient to allow distributed execution for inserts
Internal	Fix Driver Loading in Zeppelin where it fails initially
Internal	Fix a problem while reading orc byte stream
Internal	Disable dictionary cache for hbase master and spark executor
Internal	Fix Orc Partition Pruning
Internal	Making sure schema is ejected from the cache correctly
Internal	Disable Spark block cache and fix broadcast costing
Internal	Fixing ClosedConnectionException
Internal	Clean up backup endpoint to avoid hang
Internal	Update the error message when partial record is found
Internal	Suppress false constraint violation during retry
Internal	Avoid deleting a nonexist snapshot
Internal	Keep the column indexes zero based for new orc stats collection job
Internal	Correct a query to find indexes of a table
Internal	Spark job has problems renewing a kerberos ticket
Internal	Support ColumnPosition in GroupBy list
Internal	Fix wrong result for broadcast with implicit cast from int to numeric type
Internal	Fix limit on multiple partitions on Spark

Splice Patch Release 2.5.0.1729 01-Aug-17

JIRA ID	Description
SPLICE-77	order by column in subquery not projected should not be resolved
SPLICE-79	generate correct insert statement to import char for bit column
SPLICE-612	fix wrong result in right outer join with expression in join condition
SPLICE-774	Reset statistics during upgrade
SPLICE-774	Wait for master to clear upgrade znode
SPLICE-1023	add more info in the message for data import error
SPLICE-1098	prevent nonnull selectivity from being 0
SPLICE-1294	Poor Costing when first part of PK is not =
SPLICE-1395	add a generic error message for import failure from S3
SPLICE-1423	clean up import error messages for bad file
SPLICE-1438	fix the explain plan issues
SPLICE-1443	Skip cutpoint that create empty partitions
SPLICE-1452	correct cardinality estimation when there is missing partition stats
SPLICE-1461	Wrap exception parsing against errors
SPLICE-1469	Set hbase.rowlock.wait.duration to 0 to avoid deadlock
SPLICE-1470	Make sure user transaction rollbacks on Spark failure
SPLICE-1473	Allow user code to load com.splicemachine.db.iapi.error
SPLICE-1478	Fixing Statement Limits
SPLICE-1497	Add flag for inserts to skip conflict detection
SPLICE-1526	Handle CodecPool manually to avoid leaking memory
SPLICE-1533	eliminate duplicates in the IN list
SPLICE-1541	fix IN-list issues with dynamic bindings and char column

JIRA ID	Description
SPLICE-1543	fix IN-list issues with dynamic bindings and char column
SPLICE-1559	bulkImportDirectory is case sensitive
SPLICE-1582	Apply memory limit on consecutive broadcast joins
SPLICE-1584	fix IndexOutOfBoundsException when not all column stats are collected and we try to access column stats for estimation.
SPLICE-1586	Prevent NPE when Spark job fails
SPLICE-1589	All transactions are processed by pre-created region 0
SPLICE-1601	fix wrong result for min/max/sum on empty table without groupby
SPLICE-1609	normalize row source for split_table_or_index procedure
SPLICE-1622	Return only latest version for sequences
SPLICE-1624	Load pipeline driver at RS startup
SPLICE-1628	HFile bulk load is slow to copy/move HFile to regions
SPLICE-1637	Enable compression for HFile gen in bulk loader
SPLICE-1639	Fix NPE due to Spark static initialization missing
SPLICE-1640	apply memory limit check for consecutive outer broadcast join and derived tables
SPLICE-1660	Delete Not Using Index Scan due to index columns being required for the scan.
SPLICE-1675	merge partition stats at the stats collection time
SPLICE-1682	Perform accumulator check before txn resolution (2.5)
SPLICE-1684	fix stats collection logic for ArrayIndexOutOfBoundsException in the presence of empty partition and some column stats disabled
SPLICE-1690	Merge statistics on Spark (2.5)
SPLICE-1692	Perform (anti)tombstone txn resolution only when needed
SPLICE-1696	Add ScanOperation and SpliceBaseOperation to Kryo
SPLICE-1702	refresh/resolve changes with latest master branch
SPLICE-1737	fix value outside the range of the data type INTEGER error for analyze table statement.

JIRA ID	Description
SPLICE-1749	fix delete over nestedloop join
SPLICE-1759	HBase Master generates 1.1GB/s of network bandwidth even when cluster is idle
SPLICE-1781	Fixing Object Creation on IndexTransformFunction
SPLICE-1782	Code Cleanup on BulkInsertRowIndex
SPLICE-1784	Fixing Serial Cutpoint Generation
SPLICE-1791	Make username's more specific to resolve concurrent conflicts
SPLICE-1792	BroadcastJoinMemoryLimitIT must be executed serially
SPLICE-1795	fix NullPointerException for update with expression, and uncomment test case in HdfsImport related to this bug
SPLICE-1798	Parallel Queries can fail on SPS Descriptor Update...
Internal	null checking for REGEXP_LIKE
Internal	Resubmit to Spark if we consume too many resources in control
Internal	Fix a couple issues that cause backup to hang
Internal	Backups block flushes forever if not stopped cleanly
Internal	bind select statement only once in insert into select
Internal	concatenate all iterables at once to avoid stack overflow error
Internal	fix hash join column ordering
Internal	throw BR014 for concurrent backup
Internal	fix incremental backup hang
Internal	Continue processing tables when one doesn't have a namespace
Internal	correct postSplit
Internal	disable dictionary cache for hbase master and spark executor
Internal	Making sure schema is ejected from the cache correctly
Internal	Disable Spark block cache and fix broadcast costing
Internal	Fixing ClosedConnectionException handling

JIRA ID	Description
Internal	clean up backup endpoint to avoid hang
Internal	update error message when partial record is found (2.5)
Internal	suppress false constraint violation during retry
Internal	avoid deleting a nonexist snapshot
Internal	cleanup failed backup from old build
Internal	correct a query to find indexes of a table
Internal	Spark job has problems renewing a kerberos ticket
Internal	support ColumnPosition in GroupBy list
Internal	fix wrong result for broadcast with implicit cast from int to numeric type
Internal	Fix limit on multiple partitions on Spark

For a full list of JIRA's for the Community/Open Source software, see <https://splice.atlassian.net>

Limitations and Workarounds in This Release of Splice Machine

This topic describes workarounds for known limitations in this Release of the Splice Machine Database. These can include previously unstated limitations or workarounds for problems that will be fixed in a future Release.

These are the notes and workarounds for known issues in this release:

- » [With Clauses and Temporary Tables](#)
- » [Temporary Tables and Backups](#)
- » [Natural Self Joins Not Supported](#)
- » [Columnar Screen Output Gets Truncated](#)
- » [TimeStamp Date Value Limitations](#)
- » [ToDate Function Problem With DD Designator](#)
- » [Dropping Foreign Keys](#)
- » [Compaction Queue Issue](#)
- » [Alter Table Issues](#)
- » [Default Value for Lead and Lag Functions](#)
- » [CREATE TABLE AS with RIGHT OUTER JOIN](#)
- » [Import Performance Issues With Many Foreign Key References](#)

With Clauses and Temporary Tables

You cannot currently use temporary tables in WITH clauses.

Natural Self Joins Not Supported

Splice Machine does not currently support NATURAL SELF JOIN operations.

Temporary Tables and Backups

There's a subtle issue with performing a backup when you're using a temporary table in your session: although the temporary table is (correctly) not backed up, the temporary table's entry in the system tables will be backed up. When the backup is restored, the table entries will be restored, but the temporary table will be missing.

There's a simple workaround:

1. Exit your current session, which will automatically delete the temporary table and its system table entries.
2. Start a new session (reconnect to your database).
3. Start your backup job.

Columnar Screen Output Gets Truncated

When using `Explain Plan` and other commands that generate lengthy output lines, you may see some output columns truncated on the screen.

WORKAROUND: Use the `maximumdisplaywidth=0` command to force all column contents to be displayed.

TimeStamp Date Value limitations

Dates in `TIMESTAMP` Data Type values only work correctly when limited to this range of date values:

1678-01-01 to 2261-12-31

ToDate Function Problem With DD Designator

The `TO_DATE` function currently returns the wrong date if you specify `DD` for the day field; however, specifying `dd` instead works properly.

WORKAROUND: Use `dd` instead of `DD`.

Dropping Foreign Keys

The `DROP FOREIGN KEY` clause of the `ALTER TABLE` statement is currently unavailable in Splice Machine.

WORKAROUND: Re-create the table without the foreign key constraint.

Compaction Queue Issue

We have seen a problem in which the compaction queue grows quite large after importing large amounts of data, and are investigating a solution; for now, please use the following workaround.

Run a full compaction on tables into which you have imported a large amount of data, using the `SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE` system procedure.

Alter Table Issues

Using `ALTER TABLE` against `PRIMARY KEY` columns does not currently work properly.

Default Value for Lead and Lag Functions

This release of Splice Machine features several new window functions, including LAG do not support the default value parameter that you can specify in some other implementations. We expect to add this parameter in future releases.

CREATE TABLE AS with RIGHT OUTER JOIN

There is a known problem using the CREATE TABLE AS form of the RIGHT OUTER JOIN operation. For example, the following statement currently produces a table with all NULL values:

```
CREATE TABLE t3 AS
  SELECT t1.a,t1.b,t2.c,t2.d
  FROM t1 RIGHT OUTER JOIN t2 ON t1.b = t2.c
  WITH DATA;
```

There's a simple workaround for now: create the table without inserting the data, and then insert the data; for example:

```
CREATE TABLE t3 AS
  SELECT t1.a,t1.b,t2.c,t2.d
  FROM t1 RIGHT OUTER JOIN t2 ON t1.b = t2.c
  WITH NO DATA;

INSERT INTO t3
  SELECT t1.a,t1.b,t2.c,t2.d
  FROM t1 RIGHT OUTER JOIN t2 ON t1.b = t2.c;
```

Import Performance Issues With Many Foreign Key References

The presence of many foreign key references on a table will slow down imports of data into that table.

</div> </section>

Release Notes for the Splice Machine Database-as-a-Service Product

This is a DB-Service-Only topic! [Learn about our products](#)

This topic includes any release notes that are specific to the Splice Machine *Database-as-Service* product, in these sections:

- » [Features Not Yet Available](#)
- » [Current Limitations](#)
- » [Important Notes](#)

Most of the information about changes in the Splice Machine database that underlies this product are found in the Splice Machine database release notes.

Features Not Yet Available

These features are not yet available, but will be very soon:

- » VPC Settings are not yet enabled but will be in a near future release.
- » You currently cannot cancel queries that are running through Zeppelin or JDBC tools; you can use the Spark User Interface to cancel Spark queries.

Current Limitations

These limitations exist in this release, and will be removed in the near future:

- » On a JDBC connection, individual queries or actions will time out after one hour; you can run long-running queries within a Zeppelin notebook.
- » Updating of CPU, Memory, and Disk usage graphs for clusters is currently limited: the updates are happening only intermittently.

Important Notes

These are important notes about issues you need to be aware of when using our Database Service:

- » The timestamps displayed in Zeppelin will be different than the timestamps you see in the Splice Machine Spark User Interface, depending upon your time zone.
- » Although Splice Machine backs up your database regularly, it does not back up your Zeppelin Notebook changes; please export your Notebooks regularly if you make changes.

Release Notes for the Splice Machine On-Premise Product

This is an On-Premise-Only topic! [Learn about our products](#)

This topic includes release notes that are specific to the Splice Machine *On-Premise Database* product, in these sections:

- » [Supported Platforms](#)
- » [Enterprise-only Features](#)
- » [Running the Standalone Version](#)

Most of the information about changes in the Splice Machine database that forms the basis of this product are found in the Splice Machine database release notes.

After Updating

After updating to a new release of Splice Machine, you need to update your stored statement metadata by calling these two system procedures:

```
CALL SYSCS_UTIL.SYSCS_UPDATE_METADATA_STORED_STATEMENTS();
CALL SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE();
```

Supported Platforms

The supported platforms for release 2.7 are:

- » Cloudera CDH 5.12.0, 5.8.3
- » MapR 5.2.0
- » HortonWorks HDP 2.5.5, 2.6.3

Enterprise-only Features

Some features only work on the *Enterprise Edition* of Splice Machine; they **do not** work on the Community Edition of Splice Machine. To obtain a license for the Splice Machine *Enterprise Edition*, please [Contact Splice Machine Sales](#) today.

These are the enterprise-only features in our *On-Premise Database*:

- » Backup/Restore
- » LDAP integration
- » Column-level user privileges
- » Kerberos enablement

- » Encryption at rest

Running the Standalone Version

The supported operating systems for the STANDALONE release of Splice Machine are:

- » Mac OS X (10.8 or greater)
- » Centos (6.4 or equivalent)

General Information

This page includes links to various note pages, including our glossary, our trademarks page, and a few other topic pages that you may find useful.

Links to Splice Machine Notes Pages

Link	Description
Splice Machine License	Splice Machine Software End User License Agreement web page.
Spark Overview	A brief overview of Spark.
Using our Documentation	Introduces our documentation conventions and provides navigation and search tips.
Our Documentation Examples	Describes the simple database that we use for many of the examples in our documentation.
Trademarks	A list of trademarks that are referenced in our documentation.
Glossary	Definitions for terms used in our documentation that you might not know.

Splice Machine Editions

This page summarizes the features available in the different editions of Splice Machine.



If you're using the Community Edition of Splice Machine and want to learn more about upgrading to our Enterprise or Cloud editions, please [please Contact Splice Machine Sales](#) today.

Feature Comparison

This table summarizes the features that are available in each edition of Splice Machine:

Features	Cloud Edition	Enterprise Edition	Community Edition
On-Demand Compute Nodes and Storage	✓		
Managed Backups and Restores	✓		
Integrated Zeppelin Notebooks	✓		
Splice Machine Cloud Manager	✓		
Scale-Out Architecture	✓	✓	✓
ANSI SQL	✓	✓	✓
Concurrent Acid Transactions	✓	✓	✓
OLAP and OLTP Resource Isolation	✓	✓	✓
Distributed In-Memory Joins, Aggregations, Scans, and Groupings	✓	✓	✓
Cost-Based Statistics / Query Optimizer	✓	✓	✓
Hybrid Row-based and Columnar Storage	✓	✓	✓
Compaction Optimization	✓	✓	✓
Stored Procedures, Triggers, User-Defined Functions	✓	✓	✓
Apache Kafka-enabled Streaming	✓	✓	✓
Virtual Table Interfaces	✓	✓	✓
PL/SQL Support	✓	✓	
Backup and Restore Capabilities	✓	✓	
Column Level Access Control	✓	✓	
Encryption	✓	✓	

Features	Cloud Edition	Enterprise Edition	Community Edition
Security Features, including Kerberos	✓	✓	
LDAP Support		✓	
New Releases and Maintenance Updates	✓	✓	✓

Additional Materials and Support

This table summarizes the additional materials and support that are available for each edition of Splice Machine:

Features	Cloud Edition	Enterprise Edition	Community Edition
Tutorials	✓	✓	✓
Forums	✓	✓	✓
Videos	✓	✓	✓
Online Documentation	✓	✓	✓
Community Support	✓	✓	✓
24/7 Support via Web and Phone	✓	✓	
Complimentary Account Management Services	✓	✓	
GitHub Repository	✓	✓	✓

Licensing

This table compares the pricing policies for the different editions of Splice Machine:

Edition	Pricing Policy
Cloud Edition	On compute and storage units per month basis
Enterprise Edition	On a per node per year basis
Community Edition	Free

The Splice Machine In-Memory Engine

This topic provides an overview of the Splice Machine in-memory engine, which tremendously boosts OLAP (analytical) query performance. Splice Machine use Apache Spark as our in-memory engine and automatically detects and directs OLAP queries to that engine.

This topic presents a very brief overview of Spark terminology and concepts.

NOTE: If you're not yet familiar with Spark, we recommend visiting the Apache Spark web site, spark.apache.org, to learn the basics, and for links to the official Spark documentation.

Spark Overview

Apache Spark is an open source computational engine that manages tasks in a computing cluster. Spark was originally developed at UC Berkeley in 2009, and then open sourced in 2010 as an Apache project. Spark has been engineered from the ground up for performance, exploiting in-memory computing and other optimizations to provide powerful analysis on very large data sets.

Spark provides numerous performance-oriented features, including:

- » ability to cache datasets in memory for interactive data analysis
- » abstraction
- » integration with a host of data sources
- » very fast data analysis
- » easy to use APIs for operating on large datasets, including numerous operators for transforming and manipulating data, in Java, Scala, Python, and other languages
- » numerous high level libraries, including support for machine learning, streaming, and graph processing
- » scalability to thousands of nodes

Spark applications consist of a driver program and some number of worker programs running on cluster nodes. The data sets (RDDs) used by the application are distributed across the worker nodes.

Spark Terminology

Splice Machine launches Spark queries on your cluster as Spark *jobs*, each of which consists of some number of *stages*. Each stage then runs a number of *tasks*, each of which is a unit of work that is sent to an *executor*.

The table below contains a brief glossary of the terms you'll see when using the Splice Machine Database Console:

Term	Definition
Action	A function that returns a value to the driver after running a computation on an <i>RDD</i> . Examples include <code>save</code> and <code>collect</code> functions.
Application	<p>A user program built on Spark. Each application consists of a <i>driver program</i> and a number of <i>executors</i> running on your cluster.</p> <p>An application creates <i>RDDs</i>, transforms those <i>RDDs</i>, and runs <i>actions</i> on them. These result in a directed acyclic graph (DAG) of operations, which is compiled into a set of <i>stages</i>. Each stage consists of a number of <i>tasks</i>.</p>
DAG	A Directed Acyclic Graph of the operations to run on an <i>RDD</i> .
Driver program	This is the process that's running the <code>main()</code> function of the application and creating the <code>SparkContext</code> object, which sends jobs to <i>executors</i> .
Executor	A process that is launched (by the driver program) for an <i>application</i> on a <i>worker node</i> . The executor launches <i>tasks</i> and maintains data for them.
Job	A parallel computation consisting of multiple <i>tasks</i> that gets spawned in response to a Spark <i>action</i> .
Partition	A subset of the elements in an <i>RDD</i> . Partitions define the unit of parallelism; Spark processes elements within a partition in sequence and multiple partitions in parallel.
RDD	A Resilient Distributed Dataset . This is the core programming abstraction in Spark, consisting of a fault-tolerant collection of elements that can be operated on in parallel.
Stage	A set of tasks that run in parallel. The stage creates a <i>task</i> for each partition in an <i>RDD</i> , serializes those tasks, and sends those tasks to <i>executors</i> .
Task	The fundamental unit of work in Spark; each task fetches input, executes operations, and generates output.
Transformation	A function that creates a new <i>RDD</i> from an existing <i>RDD</i> .
Worker node	A cluster node that can run application code.

See Also

- » About the Splice Machine Database Console
- » User Interface Features of the Splice Machine Database Console
- » Managing Queries with the Console

- » [Using Spark Libraries with Splice Machine](#)
- » The Apache Spark web site, spark.apache.org

Using the Splice Machine Documentation

This topic helps orient you to the Splice Machine documentation, in these sections:

- » [Splice Machine Layout](#) explains the layout of our documentation web and shows you how to use the various navigation tools to land on the pages in which you're interested.
- » [Navigating our Documentation](#) summarizes and links to all of the top-level sections in the Splice Machine documentation suite.

Splice Machine Layout

Here's an image of the top portion of our documentation screen, which is called the *topbar*:



The topbar features these elements:

- » Click the *Splice Machine Documentation* title to return you to the home page of our documentation.
- » Click the [Navigation Toggle](#) to toggle the sidebar off (to expand the width of the main content) or off.
- » The [top navigation menus](#) includes our three main menus, which link to the main topic sections.
- » A [sidebar](#) for navigating to topics within the main sections.
- » A [search bar](#) that you can use to search for topic titles within the documentation.
- » The [main content area](#), which contains the content of each topic.

Internal and External Links

Links to other pages in the documentation are shown in underlined blue.

Links to external sites are also shown in underlined blue, and include a boxed arrow symbol that indicates the link will automatically open in a separate browser tab or window. For example, [this link](#) opens the Splice Machine web home page in a separate browser tab.

Navigating our Documentation

This site includes documentation for both of our Splice Machine products. The topic sections are organized into three main categories, each of which has detailed sections and is represented by one of the menus at the top of each page (*Splice Machine*, *DB-Service Only*, *On-Premise-DB Only*). The sidebar navigation (on the left) automatically changes whenever you select a new section of the docs.

The following table summarizes the main sections of our documentation:

Menu	Section	Description
<i>Splice Machine</i>	Welcome	Information about our database: the basis of all Splice Machine products.
	Tutorials	A collection of tutorials that walk you through numerous specific tasks to help you quickly learn how to use your Splice Machine database more productively.
	SQL Reference Manual	The reference manual for the Splice Machine implementation of SQL.
	<u>Developer's Manual</u>	Topics of interest to all developers working with Splice Machine.
	Command Line Reference	The reference manual for our Splice Machine command line interface.
	Database Console Guide	An introduction to the Splice Machine Spark Database Console.
	Release Notes	Information about new features, improvements, and fixes in the current database release.
General Information	General information about our database products and our documentation.	
<i>DB-Service Only</i>	Welcome	The content in this section is specific to our cloud-managed database service product.
	Cloud Manager Guide	A guide to our Cloud Manager, which is the Dashboard from which you create, manage, and use your clusters.
	Using Zeppelin	A guide to using Zeppelin notebooks to work with your Splice Machine databases.
	Release Information	Release notes, workarounds, and other information about this release.
<i>On-Premise-DB Only</i>	Welcome	The content in this section is specific to our on-premise database product.
	Installing Splice Machine	Step-by-step instructions for installing the on-premise version of Splice Machine on compatible platforms.
	On-Premise Administration	Topics that describe the administrative tasks associated with installing, configuring, and maintaining your on-premise Splice Machine database.

Menu	Section	Description
	Best Practices and Troubleshooting	Tips for best practices and solving common problems.

About Our Documentation Examples Database

This topic describes the database that we have created to provide code examples throughout our documentation suite. We've pulled in basic seasonal statistics for two Major League Baseball teams, though all names have been changed. We're calling this our DocsExamplesDb database.

Our DocsExamplesDb database features these tables:

Table	Contains
Players	The player's name, ID, and other general information.
Salaries	The salary for each player ID for each season.
Batting	Batting statistics, per season, for each player ID.
Fielding	Fielding statistics, per season, for each player ID.
Pitching	Pitching statistics, per season, for each pitcher's player ID.

The tables were populated with data found on the Internet, primarily from the baseball-reference.com site.

Table Schemas

Our example tables are all stored in a schema named SPLICEBALL. This section describes the fields in each of our DocsExamplesDb tables.

The Players Table

The SPLICEBALL.Players table contains these columns:

Column Name	Type	Description
ID	SMALLINT	The unique player ID, assigned upon insertion.
Team	VARCHAR	The abbreviated name of the player's team.
DisplayName	VARCHAR	The name we use when displaying this player.
Position	CHAR (2)	The abbreviation for the player's main position, e.g. P, C, OF, 1B.
Birthdate	DATE	The birth date of the player.

The Salaries Table

The SPLICEBALL.Salaries table contains these columns:

Column Name	Type	Description
ID	SMALLINT	The unique player ID.
Season	SMALLINT	The season (year).
Salary	BIGINT	The player's salary for the season.

The Batting Table

The SPLICEBALL.Batting contains these columns:

Column Name	Type	Description
ID	SMALLINT	The unique player ID.
Season	SMALLINT	The season (year).
Games	SMALLINT	The number of games in which the player batted.
PlateAppearances	SMALLINT	The number of times the player made a plate appearance.
AtBats	SMALLINT	The number of official at bats.
Runs	SMALLINT	The number of runs scores.
Hits	SMALLINT	How many hits by the player.
Singles	SMALLINT	How many singles hit by the player. This value is computed by a triggered function.
Doubles	SMALLINT	How many doubles hit by the player.
Triples	SMALLINT	How many triples hit by the player.
HomeRuns	SMALLINT	How many home runs hit by the player.
RBI	SMALLINT	How many Runs Batted In by the player.
StolenBases	SMALLINT	How many bases the player stole.

Column Name	Type	Description
CaughtStealing	SMALLINT	How many times the player was caught attempting to steal a base.
Walks	SMALLINT	The number of walks the player drew.
Strikeouts	SMALLINT	The number of times the player walked.
DoublePlays	SMALLINT	How many times the player hit into a double play.
HitByPitches	SMALLINT	The number of times the player was hit by a pitch.
SacrificeHits	SMALLINT	The number of sacrifice bunts the player hit.
SacrificeFlies	SMALLINT	The number of sacrifice flies the player hit.
IntentionalWalks	SMALLINT	The number of intentional walks issued to the player.
Average	DECIMAL	<p>The player's batting average.</p> <p>This value is computed by a triggered function.</p>
TotalBases	SMALLINT	<p>The total number of bases for the player.</p> <p>This value is computed by a triggered function.</p>
OnBasePercentage	DECIMAL	<p>The percentage of times the player reached base.</p> <p>This value is computed by a triggered function.</p>
Slugging	DECIMAL	<p>The slugging average of the player.</p> <p>This value is computed by a triggered function.</p>
OnBasePlusSlugging	DECIMAL	<p>The OPS for the player.</p> <p>This value is computed by a triggered function.</p>

The Fielding Table

The SPLICEBALL.Fielding table contains these columns:

Column Name	Type	Description
ID	SMALLINT	The unique player ID.
Season	SMALLINT	The season (year).
FldGames	SMALLINT	How many games the player was in the field for.
Chances	SMALLINT	The number of fielding chances the player had.
Putouts	SMALLINT	The number of putouts the player had.
Assists	SMALLINT	The number of assists the player had.
Errors	SMALLINT	The number of errors committed by the player.
FldDoublePlays	SMALLINT	The number of doubles plays in which the player was involved in as a fielder.
Percentage	DECIMAL	The percentage of opportunities for outs that the player successfully completed.
TZAboveAverage	SMALLINT	A fielding metric: total zone runs above average for his position.
TZAboveAveragePer1200	SMALLINT	Total zone runs extrapolated for 1200 innings.
RunsSaved	SMALLINT	The number of runs saved in the field by the player.
RunsSavedAboveAvg	SMALLINT	The number of runs saved by the player over the average number saved for his position.
RangeFactorPerNine	DECIMAL	A fielding metric that evaluates the average number of putouts and assists per nine innings,
RangeFactorPerGame	DECIMAL	A fielding metric that evaluates the average number of putouts and assists per game played,
PassedBalls	SMALLINT	The number of passed balls for catchers.
WildPitches	SMALLINT	The number of wild pitches for pitchers.
FldStolenBases	SMALLINT	The number of stolen bases given up by a pitcher or catcher.
FldCaughtStealing	SMALLINT	The number of players caught stealing by a pitcher or catcher.

Column Name	Type	Description
FldCaughtStealingPercent	DECIMAL	For catchers and pitchers, the percentage of attempted stolen bases that were successful.
FldLeagueCaughtStealingPercent	DECIMAL	For pitchers and catchers, the league average percentage of attempted stolen bases that were successful.
Pickoffs	SMALLINT	For pitchers and catchers, the number of runners picked off.
FldInnings	DECIMAL	The number of innings in which the player was in the field.

The Pitching Table

The SPLICEBALL.Pitching table contains these fields:

Column Name	Type	Description
ID	SMALLINT	The unique player ID.
Season	SMALLINT	The season (year).
Wins	SMALLINT	How many games the pitcher won.
Losses	SMALLINT	How many games the pitcher lost.
Games	SMALLINT	The number of games in which the pitcher appeared.
GamesStarted	SMALLINT	The number of games the pitcher started.
GamesFinished	SMALLINT	The number of games the pitcher finished.
CompleteGames	SMALLINT	The number of complete games by the pitcher.
Shutouts	SMALLINT	The number of shutout games thrown by the pitcher.
Saves	SMALLINT	The number of games saved by the pitcher.
Innings	DECIMAL	The number of innings pitched.
Hits	SMALLINT	The number of hits given up by the pitcher.
Runs	SMALLINT	The number of runs give up by the pitcher.

Column Name	Type	Description
EarnedRuns	SMALLINT	The number of earned runs give up by the pitcher.
HomeRuns	SMALLINT	How many homeruns the pitcher gave up.
Walks	SMALLINT	How many walks the pitcher issued.
IntentionalWalks	SMALLINT	How many intentional walks the pitcher issued.
Strikeouts	SMALLINT	How many batters the pitchers struck out.
HitBatters	SMALLINT	How many batters the pitcher hit with a pitch.
Balks	SMALLINT	How many balks the pitcher committed.
WildPitches	SMALLINT	How many wild pitches were thrown by the pitcher.
BattersFaced	SMALLINT	The number of batters faced by the pitcher.
FieldingIndependent	DECIMAL	A metric (FIP) for pitchers that determines the quality of a pitcher's performance by eliminating plate appearance outcomes that involve defensive play.
ERA	DECIMAL	<p>The pitcher's earned run average.</p> <p>This value is computed by a triggered function.</p>
WHIP	DECIMAL	<p>The number of walks and hits per inning pitched by the player.</p> <p>This value is computed by a triggered function.</p>
HitsPerNine	DECIMAL	<p>The number of hits per nine innings pitched by the player.</p> <p>This value is computed by a triggered function.</p>
HomeRunsPerNine	DECIMAL	<p>The number of home runs per nine innings pitched by the player.</p> <p>This value is computed by a triggered function.</p>
WalksPerNine	DECIMAL	<p>The number of walks per nine innings pitched by the player.</p> <p>This value is computed by a triggered function.</p>

Column Name	Type	Description
StrikeoutsPerNine	DECIMAL	<p>The number of strikeouts per nine innings pitched by the player.</p> <p>This value is computed by a triggered function.</p>
StrikeoutsToWalks	DECIMAL	<p>The ratio of strikeouts to walks thrown by the pitcher.</p> <p>This value is computed by a triggered function.</p>

Trademarks

The following table lists third-part trademark information for products mentioned in this documentation suite:

Trademark Holder	Trademarks
Amazon Web Services, Inc.	AWS and Amazon Elastic Compute Cloud are trademarks of Amazon Web Services, Inc.
Apache Software Foundation	Apache, Apache Derby, Apache Spark, HBase, and Hive are trademarks of the Apache Software Foundation. Hadoop is a registered trademark of the Apache Software Foundation.
Apple Computer, Inc.	Mac OS and OS X are registered trademarks of Apple Computer, Inc.
Canonical Limited	Ubuntu is a registered trademark of Canonical Limited.
Cloudera, Inc.	Cloudera is a trademark of Cloudera Corporation.
DbVis Software AB	DbVisualizer is trademark of DbVis Software AB.
EasySoft Limited	EasySoft is a trademark of EasySoft Limited.
Hortonworks, Inc.	Hortonworks and HDP are registered trademarks of Hortonworks, Inc.
Linus Torvalds	Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.
MapR Technologies, Inc.	MapR is a registered trademark of MapR Technologies, Inc.
Microsoft Corporation	Windows is a registered trademark of Microsoft Corporation.
MongoDB, Inc.	MongoDb is a registered trademark of MongoDB, Inc.
Oracle Corporation	MySQL is a trademark of Oracle Corporation or its affiliates. Oracle, Java, and JDBC are registered trademarks of Oracle Corporation or its affiliates.
Pentaho, Inc.	Pentaho is a registered trademark of Pentaho, Inc.
Red Hat, Inc.	Red Hat, Red Hat Linux, Red Hat Enterprise Linux, and CentOS are registered trademarks of Red Hat, Inc
Tableau Software, Inc.	Tableau and Tableau Software are registered trademarks of Tableau Software.
The Open Group	UNIX is a registered trademark of The Open Group in the U.S. and other countries.

Glossary

Term	Definition
ACID Transactions	<p>ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably. In the context of databases, a single logical operation on the data is called a transaction.</p> <ul style="list-style-type: none"> » <i>Atomicity</i> means that if one part of the transaction fails, the entire transaction fails. » <i>Consistency</i> ensures that any transaction will bring the database from one valid state to another, which means that any data written to the database must be valid according all rules defined in the database. » <i>Isolation</i> ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed in serial order. » <i>Durability</i> means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.
Auto-sharding	The database is automatically and transparently partitioned (sharded) across low cost commodity nodes, allowing scale-out of read and write queries, without requiring changes to the application.
BI Tools	Business Intelligence Tools
CDH	Clouderas <i>Cloudera Distribution Including Apache Hadoop</i> , a popular Hadoop platform.
Column-Oriented Data Model	A model for storing data in a database as sections of columns, rather than as rows of data. In a column-oriented database, all of the values in a column are serialized together.
Concurrency	The ability for multiple users to access data at the same time.
CRM	Customer Relationship Management
Cross-table, cross-row transactions	A transaction (a group of SQL statements) can modify multiple rows (cross-row) in multiple tables (across tables).
CRUD	Create, Read, Update, Delete. The four basic functions of persistent storage.
DAG	Directed Acyclic Graph. A directed graph with no directed cycles, meaning that no path through the graph loops back to its starting point. DAGs are used for various computational purposes, including query optimization in some databases.
Database Statistics	A form of dynamic metadata that assists the query optimizer in making better decisions by tracking distribution of values in indexes and/or columns.

Term	Definition
<i>Database Transaction</i>	A sequence of database operations performed as a single logical unit of work.
<i>ERP</i>	Enterprise Resource Planning is business management software that a company can use to collect, store, manage and interpret data from many business activities.
<i>Foreign Key</i>	A column or columns in one table that references a column (typically the primary key column) of another table. Foreign keys are used to ensure referential integrity.
<i>Full join support</i>	Databases use join operations to combine fields from multiple tables by using values common to each. Full join support means that the Database Management System supports all five ANSI-standard types of join operations: Inner join, left outer join, right outer join, full outer join, and cross join.
<i>Hadoop</i>	An Apache open source software project that enables the distributed processing of large data sets across clusters of commodity servers. It is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance.
<i>HBase</i>	A column-oriented database management system that is part of the Apache Hadoop framework and runs on top of HDFS.
<i>HCatalog</i>	Apache HCatalog is a table and storage management layer for Hadoop that enables users with different data processing tools to more easily read and write data on the grid.
<i>HDFS</i>	Hadoop Distributed File System . A distributed file system that stores data on commodity hardware and is part of the Apache Hadoop framework. It links together the file systems on many local nodes to make them into one big file system.
<i>HDP</i>	Hortonworks Data Platform includes Apache Hadoop and is used for storing, processing, and analyzing large volumes of data. The platform is designed to deal with data from many sources and formats.
<i>HIVE</i>	Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis.
<i>HR</i>	Human Resources .
<i>JDBC</i>	Java DataBase Connection . An API specification for connecting with databases using programs written in Java.
<i>JSON</i>	An open standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs. It is used primarily to transmit data between a server and web applications, as an alternative to XML.
<i>JVM</i>	Java Virtual Machine . The code execution component of the Java platform.
<i>Key-Value Data Model</i>	A fundamental and open-ended data model that allows for extension without modifying existing data. Data is represented in pairs: name (or key) and a value that is associated with that name. Also known as key-value pair, name-value pair, field-value pair, and attribute-value pair.

Term	Definition
<i>Map Reduce</i>	<p>MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. MapReduce programs typically include these steps:</p> <ol style="list-style-type: none"> 1. Each worker node applies the <code>Map()</code> function to filter and sort local data. 2. Worker nodes redistribute (shuffle) data based on output keys produced by the <code>Map()</code> step, so that all data belonging to one key is located on the same node. 3. Worker nodes process each group of output data in parallel to produce results. 4. The MapReduce system collects the results and sorts them to produce the final outcome.
<i>MapR</i>	<p><i>MapR</i> is a complete distribution for Apache Hadoop that packages more than a dozen projects from the Hadoop ecosystem to provide a broad set of big data capabilities.</p>
<i>Multi-partition transactions</i>	<p>A database, like Splice Machine, that allows transactions for a table distributed as multiple partitions across multiple nodes in a cluster.</p>
<i>MVCC</i>	<p>MultiVersion Concurrency Control is a method used to control concurrent access to a database. Concurrency control is needed to bypass the potential problem of someone viewing (reading) a data value while another is writing to the same value.</p>
<i>MySQL</i>	<p>An open source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL).</p>
<i>NewSQL</i>	<p>NewSQL is a class of modern relational database management systems that seek to provide the same scalable performance of NoSQL systems for online transaction processing (OLTP) read-write workloads while still maintaining the ACID guarantees of a traditional database system.</p>
<i>NoSQL</i>	<p>A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.</p>
<i>ODBC</i>	<p>Open DataBase Connectivity. An open standard API for accessing database management systems, designed to be independent of any specific database or operating system.</p>
<i>OLAP</i>	<p>OnLine Analytical Processing. An approach to quickly answering multi-dimensional analytical queries in the Business Intelligence world. OLAP tools allow users to analyze multidimensional data interactive from multiple perspectives.</p>
<i>OLTP</i>	<p>OnLine Transaction Processing. A class of information processing systems that facilitate and manage transaction-oriented applications.</p>
<i>Query Optimizer</i>	<p>A critical database management system (DBMS) component that analyzes SQL queries and determines efficient execution mechanisms, known as query plans. The optimizer typically generates several plans and then selects the most efficient plan to run the query.</p>

Term	Definition
<i>Referential Integrity</i>	A property of data that requires every value of one column in a table to exist as a value of another column in a different (or the same) table. This term is generally used to describe the function of foreign keys.
<i>Relational Data Model</i>	<p>The model, developed by E.F. Codd, upon which relational database are based. Relational tables have these properties:</p> <ul style="list-style-type: none"> » Data is presented as a collection of relations » Each relation is depicted as a table » Columns are attributes that belong to the entity modeled by the table » Each row represents a single entity (a record) » Every table has a set of attributes, a key, that unique identifies each entity
<i>REST</i>	RE presentational S tate T ransfer. A simple, stateless, client-server protocol use for networked applications, which uses the HTTP requests to communicate among machines. RESTful applications use HTTP requests to post (create or update) data, to read data, and to delete data. Collectively, these are known as CRUD operations.
<i>Rollback</i>	An operation that returns the database to some previous state, typically used for recovering from database server crashes: by rolling back any transaction that was active at the time of the crash, the database is restored to a consistent state.
<i>Scale Out</i>	A database architecture that doesn't rely on a single controller and scales by adding processing power coupled with additional storage.
<i>Scale Up</i>	An architecture that uses a fixed controller resource for all processing. Scaling capacity happens by adding processing power to the controller or (eventually) upgrading to a new (and expensive) controller.
<i>Sharding</i>	Horizontal partitioning in a database: that the data is split among multiple machines while ensuring that the data is always accessed from the correct place. See Auto-sharding.
<i>Spark</i>	Apache Spark is an open-source cluster computing framework that uses in-memory primitives to reduce storage access and boost database performance. Spark allows user applications to load data into a cluster's memory and repeatedly query that data.
<i>Trigger</i>	A database trigger is code that is run automatically in response to specific events on specific tables or views in your database. Triggers are typically configured to maintain data integrity, such as ensuring that an updated value is in range.
<i>YARN</i>	Y et A nother R esource N egotiator. YARN assigns CPU, memory, and storage to applications running on a Hadoop cluster, and enables application frameworks other than MapReduce (like Spark) to run on Hadoop.

Term	Definition
<i>ZooKeeper</i>	Part of the Apache Hadoop framework, ZooKeeper provides a centralized infrastructure and services that enable synchronization across a cluster. ZooKeeper maintains common objects needed in large cluster environments. Examples of these objects include configuration information, hierarchical naming space, and so on. Applications can leverage these services to coordinate distributed processing across large clusters.