Table of Contents

Splice Machine Splice Machine Documentation

Splice Machine Documentation
  Welcome........................................................................................................................................................................ 1

About Splice Machine
  Getting Started................................................................................................................................................................ 4

Splice Machine Release Notes
  Introduction....................................................................................................................................................................... 10
  Known Issues and Workarounds.................................................................................................................................... 12
  Patch Release 2.7.0.1931........................................................................................................................................... 16
  Patch Release 2.7.0.1930........................................................................................................................................... 17
  Patch Release 2.7.0.1929........................................................................................................................................... 18
  Patch Release 2.7.0.1928........................................................................................................................................... 19
  Patch Release 2.7.0.1927........................................................................................................................................... 20
  Patch Release 2.7.0.1926........................................................................................................................................... 21
  Patch Release 2.7.0.1925........................................................................................................................................... 22
  Patch Release 2.7.0.1924........................................................................................................................................... 23
  Patch Release 2.7.0.1923........................................................................................................................................... 24
  Patch Release 2.7.0.1922........................................................................................................................................... 25
  Patch Release 2.7.0.1921........................................................................................................................................... 26
  Patch Release 2.7.0.1920........................................................................................................................................... 27
  Patch Release 2.7.0.1919........................................................................................................................................... 28
  Patch Release 2.7.0.1918........................................................................................................................................... 29
  Patch Release 2.7.0.1917........................................................................................................................................... 30
  Patch Release 2.7.0.1916........................................................................................................................................... 31
  Patch Release 2.7.0.1915........................................................................................................................................... 32
  Patch Release 2.7.0.1914........................................................................................................................................... 33
  Patch Release 2.7.0.1913........................................................................................................................................... 34
  Patch Release 2.7.0.1912........................................................................................................................................... 35
  Patch Release 2.7.0.1911........................................................................................................................................... 36
  Patch Release 2.7.0.1910........................................................................................................................................... 37
  Patch Release 2.7.0.1909........................................................................................................................................... 38
  Patch Release 2.7.0.1908........................................................................................................................................... 39
Using the DataSource........................................................................................................................................... 173
API Methods......................................................................................................................................................... 177
Running Apps...................................................................................................................................................... 187
Notebook Example .......................................................................................................................................... 192

DB-as-Service Product Info

DB-as-Service Welcome...................................................................................................................................... 195
About our DB-as-Service................................................................................................................................... 196
Service Overview................................................................................................................................................ 198
User Interface Overview.................................................................................................................................... 200

Cloud Manager Guide

Introduction.......................................................................................................................................................... 201
Registering with Cloud Manager....................................................................................................................... 203
Logging into the Cloud Manager....................................................................................................................... 205
Creating Your Cluster..................................................................................................................................... 206
Exploring Your Dashboard................................................................................................................................. 217
Managing a Cluster.......................................................................................................................................... 219
Managing Your Account................................................................................................................................... 225
Reviewing Event Notifications......................................................................................................................... 231
Enabling Azure Active Directory..................................................................................................................... 233

Using Zeppelin

Introduction.......................................................................................................................................................... 238
Getting Started with Zeppelin.......................................................................................................................... 239
Zeppelin Usage Notes...................................................................................................................................... 247
A Simple Tutorial............................................................................................................................................. 249

Developer Quick Start

Release Notes..................................................................................................................................................... 254

Developers Guide / Tutorials

Introduction.......................................................................................................................................................... 256

Accessing Data in the Cloud

Introduction.......................................................................................................................................................... 261
Setting up S3 Access.......................................................................................................................................... 262
Uploading Data to S3.......................................................................................................................................... 271
Using Azure Storage.......................................................................................................................................... 279

Connecting BI Tools

Introduction.......................................................................................................................................................... 284
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connecting with JDBC</td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>300</td>
</tr>
<tr>
<td>Connecting via JDBC with Java</td>
<td>302</td>
</tr>
<tr>
<td>Connecting via JDBC with JRuby</td>
<td>305</td>
</tr>
<tr>
<td>Connecting via JDBC with Jython</td>
<td>308</td>
</tr>
<tr>
<td>Connecting via JDBC with Python</td>
<td>311</td>
</tr>
<tr>
<td>Connecting via JDBC with R</td>
<td>312</td>
</tr>
<tr>
<td>Connecting via JDBC with Scala</td>
<td>314</td>
</tr>
<tr>
<td>Connecting via JDBC with Angular/NodeJS</td>
<td>318</td>
</tr>
<tr>
<td>JDBC Connections with Kerberos</td>
<td>319</td>
</tr>
<tr>
<td>Connecting with ODBC</td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>320</td>
</tr>
<tr>
<td>Installing our ODBC Driver</td>
<td>321</td>
</tr>
<tr>
<td>ODBC Connections with Kerberos</td>
<td>339</td>
</tr>
<tr>
<td>Connecting via ODBC with Python</td>
<td>340</td>
</tr>
<tr>
<td>Connecting via ODBC with C</td>
<td>343</td>
</tr>
<tr>
<td>Using Attunity Replicate</td>
<td></td>
</tr>
<tr>
<td>Exporting MySQL to Splice Machine</td>
<td>345</td>
</tr>
<tr>
<td>Exporting via ODBC to Splice Machine</td>
<td>348</td>
</tr>
<tr>
<td>Functions and Stored Procedures</td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>351</td>
</tr>
<tr>
<td>Writing Functions and Stored Procedures</td>
<td>355</td>
</tr>
<tr>
<td>Storing/Updating Functions and Procs</td>
<td>364</td>
</tr>
<tr>
<td>Examples</td>
<td>367</td>
</tr>
<tr>
<td>Fundamentals</td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>372</td>
</tr>
<tr>
<td>Compaction and Vacuuming</td>
<td>373</td>
</tr>
<tr>
<td>Running Transactions</td>
<td>375</td>
</tr>
<tr>
<td>Using Database Triggers</td>
<td>386</td>
</tr>
<tr>
<td>Using External Tables</td>
<td>390</td>
</tr>
<tr>
<td>Using Foreign Keys</td>
<td>394</td>
</tr>
<tr>
<td>Using Spark Libraries</td>
<td>396</td>
</tr>
<tr>
<td>Using Temporary Tables</td>
<td>404</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Using the Virtual Table Interface</td>
<td>407</td>
</tr>
<tr>
<td>Using Window Functions</td>
<td>417</td>
</tr>
<tr>
<td>Working with Date and Time Values</td>
<td>428</td>
</tr>
<tr>
<td><strong>Indexing</strong></td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>434</td>
</tr>
<tr>
<td>Indexing Large Tables</td>
<td>435</td>
</tr>
<tr>
<td><strong>On-Premise-Database Authentication</strong></td>
<td></td>
</tr>
<tr>
<td>About Database Authentication</td>
<td>443</td>
</tr>
<tr>
<td>Using NATIVE Authentication</td>
<td>445</td>
</tr>
<tr>
<td>Using LDAP Authentication</td>
<td>446</td>
</tr>
<tr>
<td>Using Kerberos with Splice Machine</td>
<td>450</td>
</tr>
<tr>
<td><strong>On-Premise-DB Developer Topics</strong></td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>452</td>
</tr>
<tr>
<td>Connecting Through HAProxy</td>
<td>453</td>
</tr>
<tr>
<td>Using HCatalog</td>
<td>459</td>
</tr>
<tr>
<td>Using the MapReduce API</td>
<td>466</td>
</tr>
<tr>
<td>Using Zeppelin</td>
<td>470</td>
</tr>
<tr>
<td>Working with HBase</td>
<td>475</td>
</tr>
<tr>
<td><strong>Securing Your Database</strong></td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>478</td>
</tr>
<tr>
<td>Authorizing Users and Roles</td>
<td>479</td>
</tr>
<tr>
<td>Securing Connections with SSL/TLS</td>
<td>483</td>
</tr>
<tr>
<td>User/Role Permissions Summary</td>
<td>492</td>
</tr>
<tr>
<td>Using Ranger</td>
<td>496</td>
</tr>
<tr>
<td>Windows Access with Kerberos</td>
<td>503</td>
</tr>
<tr>
<td><strong>Splice*Plus (PL/SQL)</strong></td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>508</td>
</tr>
<tr>
<td>Using Splice*Plus</td>
<td>509</td>
</tr>
<tr>
<td>The Splice*Plus Language</td>
<td>513</td>
</tr>
<tr>
<td><strong>Tuning and Debugging</strong></td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>518</td>
</tr>
<tr>
<td>Debugging</td>
<td>519</td>
</tr>
<tr>
<td>Explain Plan Examples</td>
<td>521</td>
</tr>
<tr>
<td>Logging</td>
<td>530</td>
</tr>
<tr>
<td>Using Explain Plan</td>
<td>534</td>
</tr>
<tr>
<td>Optimizing Queries</td>
<td>537</td>
</tr>
<tr>
<td>Using Snapshots</td>
<td>544</td>
</tr>
</tbody>
</table>
### Splice Machine ML Manager

**ML Manager Overview** ................................................................. 708
**Using ML Manager** ................................................................. 710

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit command</td>
<td>650</td>
</tr>
<tr>
<td>Execute command</td>
<td>651</td>
</tr>
<tr>
<td>Explain Plan command</td>
<td>653</td>
</tr>
<tr>
<td>Export command</td>
<td>654</td>
</tr>
<tr>
<td>Export_Binary command</td>
<td>657</td>
</tr>
<tr>
<td>Prepare command</td>
<td>659</td>
</tr>
<tr>
<td>Release Savepoint command</td>
<td>660</td>
</tr>
<tr>
<td>Remove Command</td>
<td>663</td>
</tr>
<tr>
<td>Rollback command</td>
<td>664</td>
</tr>
<tr>
<td>Rollback to Savepoint command</td>
<td>665</td>
</tr>
<tr>
<td>Savepoint command</td>
<td>667</td>
</tr>
</tbody>
</table>

**Commands Only Available via sqlshell.sh**

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect command</td>
<td>669</td>
</tr>
<tr>
<td>Describe command</td>
<td>671</td>
</tr>
<tr>
<td>Disconnect command</td>
<td>673</td>
</tr>
<tr>
<td>ElapsedTime command</td>
<td>674</td>
</tr>
<tr>
<td>Exit command</td>
<td>675</td>
</tr>
<tr>
<td>Help command</td>
<td>676</td>
</tr>
<tr>
<td>MaximumDisplayWidth command</td>
<td>678</td>
</tr>
<tr>
<td>Run command have a run command</td>
<td>679</td>
</tr>
<tr>
<td>Set Connection command</td>
<td>680</td>
</tr>
<tr>
<td>Set Session_Property command</td>
<td>681</td>
</tr>
<tr>
<td>Show Connections command</td>
<td>684</td>
</tr>
<tr>
<td>Show Create Table command</td>
<td>685</td>
</tr>
<tr>
<td>Show Functions command</td>
<td>686</td>
</tr>
<tr>
<td>Show Indexes command</td>
<td>687</td>
</tr>
<tr>
<td>Show PrimaryKeys command</td>
<td>689</td>
</tr>
<tr>
<td>Show Procedures command</td>
<td>690</td>
</tr>
<tr>
<td>Show Roles command</td>
<td>695</td>
</tr>
<tr>
<td>Show Schemas command</td>
<td>696</td>
</tr>
<tr>
<td>Show Synonyms command</td>
<td>697</td>
</tr>
<tr>
<td>Show Tables command</td>
<td>698</td>
</tr>
<tr>
<td>Show Views command</td>
<td>700</td>
</tr>
</tbody>
</table>

**Scripting splice> Commands** ..................................................................................................................... 701
**Using RLWrap** ............................................................................................................................................. 706

---

**Splice Machine Documentation**

vii
## SQL Reference Manual

### Introduction

- **ML Manager API**
  - Page: 732

- **Built-in Functions (A to F)**
  - **ABS**
    - Page: 740
  - **ACOS**
    - Page: 741
  - **ADD_MONTHS**
    - Page: 743
  - **ASIN**
    - Page: 746
  - **ATAN**
    - Page: 748
  - **ATAN2**
    - Page: 750
  - **AVG**
    - Page: 752
  - **BIGINT**
    - Page: 756
  - **CAST**
    - Page: 758
  - **CEIL**
    - Page: 764
  - **CHAR**
    - Page: 766
  - **CHR**
    - Page: 769
  - **COALESCE**
    - Page: 771
  - **CONCAT (||) Operator**
    - Page: 773
  - **COS**
    - Page: 775
  - **COSH**
    - Page: 777
  - **COT**
    - Page: 779
  - **COUNT**
    - Page: 781
  - **CURRENT SCHEMA**
    - Page: 784
  - **CURRENT_DATE**
    - Page: 785
  - **CURRENT_ROLE**
    - Page: 787
  - **CURRENT_TIME**
    - Page: 789
  - **CURRENT_TIMESTAMP**
    - Page: 791
  - **CURRENT_USER**
    - Page: 793
  - **DATE**
    - Page: 794
  - **DAY**
    - Page: 797
  - **DEGREES**
    - Page: 800
  - **DENSE_RANK**
    - Page: 802
  - **DIGITS**
    - Page: 808
  - **DOUBLE**
    - Page: 810
  - **EXP**
    - Page: 812
  - **EXTRACT**
    - Page: 813
  - **FIRST_VALUE**
    - Page: 818
  - **FLOOR**
    - Page: 820
### Built-in Functions (G to L)

- **GROUP_USER** .......................................................... 821
- **HOUR** ........................................................................ 822
- **INITCAP** ................................................................. 823
- **INSTR** ........................................................................ 825
- **INTEGER** .............................................................. 827
- **LAG** ........................................................................ 829
- **LAST_DAY** ........................................................... 832
- **LAST_VALUE** ......................................................... 835
- **LCASE or LOWER** ................................................ 837
- **LEAD** ........................................................................ 840
- **LEFT** ......................................................................... 844
- **LENGTH** ..................................................................... 845
- **LN (or LOG)** .......................................................... 848
- **LOCATE** ................................................................. 849
- **LOG10** .................................................................... 851
- **LTRIM** ....................................................................... 852

### Built-in Functions (M to R)

- **MAX** ........................................................................ 854
- **MIN** .......................................................................... 857
- **MINUTE** ................................................................. 860
- **MOD** .......................................................................... 861
- **MONTH** ................................................................. 865
- **MONTHNAME** .......................................................... 867
- **MONTH_BETWEEN** .................................................. 870
- **NEXT_DAY** ............................................................. 872
- **NOW** .......................................................................... 875
- **NULLIF** ...................................................................... 877
- **NVL** ........................................................................... 881
- **PI** ................................................................................ 883
- **QUARTER** ............................................................... 884
- **RADIANS** ............................................................... 887
- **RAND** .......................................................................... 889
- **RANDOM** ............................................................... 890
- **RANK** .......................................................................... 891
- **REGEXP_LIKE** ....................................................... 897
- **REPEAT** ...................................................................... 899
- **REPLACE** ..................................................................... 901
- **RIGHT** ......................................................................... 903
Built-in Functions (S to Z)
SECOND........................................................................................................................................................................... 913
SESSION_USER.......................................................................................................................................................... 914
SIGN............................................................................................................................................................................... 915
SIN.................................................................................................................................................................................. 916
SINH.............................................................................................................................................................................. 918
SMALLINT..................................................................................................................................................................... 920
SQRT............................................................................................................................................................................. 922
STDDEV_POP............................................................................................................................................................. 926
STDDEV_SAMP........................................................................................................................................................... 928
STRIP............................................................................................................................................................................ 930
SUBSTR......................................................................................................................................................................... 934
SUM................................................................................................................................................................................ 936
TAN................................................................................................................................................................................. 939
TANH............................................................................................................................................................................. 941
TIME............................................................................................................................................................................... 943
TIMESTAMP................................................................................................................................................................. 945
TIMESTAMPADD.......................................................................................................................................................... 948
TIMESTAMPDIFF.......................................................................................................................................................... 951
TINYINT.......................................................................................................................................................................... 954
TO_CHAR....................................................................................................................................................................... 956
TO_DATE......................................................................................................................................................................... 958
TO_TIME....................................................................................................................................................................... 966
TO_TIMESTAMP........................................................................................................................................................... 973
TRIM............................................................................................................................................................................... 981
TRUNCATE..................................................................................................................................................................... 984
UCASE or UPPER........................................................................................................................................................ 990
USER.............................................................................................................................................................................. 993
VARCHAR...................................................................................................................................................................... 994
WEEK............................................................................................................................................................................ 996
YEAR............................................................................................................................................................................. 998

Clauses
Introduction................................................................................................................................................................. 1000
CONSTRAINT............................................................................................................................................................... 1002
<table>
<thead>
<tr>
<th>Identifier</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONG VARCHAR</td>
<td>1104</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>1105</td>
</tr>
<tr>
<td>REAL</td>
<td>1107</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>1109</td>
</tr>
<tr>
<td>TEXT</td>
<td>1110</td>
</tr>
<tr>
<td>TIME</td>
<td>1112</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>1114</td>
</tr>
<tr>
<td>TINYINT</td>
<td>1118</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>1119</td>
</tr>
</tbody>
</table>

**Identifiers**

- **Introduction**: 1120
- **Identifier Types**: 1122
- **Identifier Syntax**: 1126

**Join Operations**

- **Introduction**: 1129
- **About Join Operations**: 1130
- **CROSS JOIN**: 1132
- **INNER JOIN**: 1134
- **LEFT OUTER JOIN**: 1138
- **NATURAL JOIN**: 1140
- **RIGHT OUTER JOIN**: 1142

**Queries**

- **Introduction**: 1144
- **Query**: 1145
- **Scalar Subquery**: 1147
- **Table Subquery**: 1148

**Statements**

- **Introduction**: 1150
- **ALTER TABLE**: 1154
- **CALL (Procedure)**: 1165
- **column-definition**: 1166
- **CREATE EXTERNAL TABLE**: 1168
- **CREATE FUNCTION**: 1173
- **CREATE INDEX**: 1179
- **CREATE PROCEDURE**: 1187
- **CREATE ROLE**: 1192
- **CREATE SCHEMA**: 1194
- **CREATE SEQUENCE**: 1196
### System Procedures (A to F)

- **BACKUP_DATABASE** ................................................................. 1282
- **BACKUP_SCHEMA** ................................................................. 1286
- **BACKUP_TABLE** ................................................................. 1291
- **BULK_IMPORT_HFILE** .......................................................... 1296
- **CANCEL_BACKUP** ............................................................... 1302
- **CHECK_TABLE** ................................................................. 1303
- **COLLECT_SCHEMA_STATISTICS** ......................................... 1306
System Procedures (G to L)
GET_ACTIVE_SERVERS................................................................. 1343
GET_ALL_PROPERTIES............................................................. 1344
GET_CURRENT_TRANSACTION..................................................... 1348
GET_GLOBAL_DATABASE_PROPERTY function........................... 1349
GET_ENCODED_REGION_NAME.................................................. 1351
GET_LOGGERS........................................................................ 1354
GET_LOGGER_LEVEL................................................................. 1360
GET_REGIONS......................................................................... 1362
GET_REGIONS_SERVER_STATS_INFO.......................................... 1366
GET_REQUESTS....................................................................... 1367
GET_RUNNING_OPERATIONS.................................................... 1368
GET_SCHEMA_INFO................................................................. 1370
GET_SESSION_INFO................................................................. 1372
GET_START_KEY.................................................................... 1374
GET_VERSION_INFO................................................................. 1376
GET_WRITE_INTAKE_INFO......................................................... 1377
IMPORT_DATA........................................................................ 1379
INSTALL_JAR......................................................................... 1384
INVALIDATE_DICTIONARY_CACHE............................................ 1386
INVALIDATE_GLOBAL_DICTIONARY_CACHE............................... 1387
INVALIDATE_STORED_STATEMENTS........................................... 1388
KILL_OPERATION.................................................................... 1389

System Procedures (M to R)

COMPACT_REGION.................................................................... 1309
COMPUTE_SPLIT_KEY................................................................ 1312
CREATE_USER......................................................................... 1317
DELETE_BACKUP..................................................................... 1320
DELETE_OLD_BACKUPS............................................................. 1322
DELETE_REGION..................................................................... 1324
DELETE_SNAPSHOT.................................................................. 1328
DISABLE_COLUMN_STATISTICS............................................... 1329
DROP_SCHEMA_STATISTICS...................................................... 1331
DROP_USER............................................................................ 1333
EMPTY_GLOBAL_STATEMENT_CACHE........................................ 1335
EMPTY_STATEMENT_CACHE...................................................... 1337
ENABLE_COLUMN_STATISTICS.................................................. 1339
ENABLE.ENTERPRISE.............................................................. 1341

System Tables

SYSALIASES system table

Introduction

System Procedures (S to Z)

SET_GLOBAL_DATABASE_PROPERTY .......................................................... 1438
SET_LOGGER_LEVEL ................................................................................. 1440
SET_PURGE_DELETED_ROWS ................................................................. 1442
SET_STATS_EXTRAPOLATION ................................................................. 1443
SHOW_CREATE_TABLE ............................................................................ 1445
SNAPSHOT_SCHEMA ............................................................................... 1447
SNAPSHOT_TABLE ................................................................................ 1448
SPLIT_TABLE_OR_INDEX ........................................................................ 1449
SPLIT_TABLE_OR_INDEX_AT_POINTS ...................................................... 1455
UPDATE_ALL_SYSTEM_PROCEDURES .................................................. 1457
UPDATE_METADATA_STORED_STATEMENTS .......................................... 1459
UPDATE_SCHEMA_OWNER ...................................................................... 1461
UPDATE_SYSTEM_PROCEDURE ............................................................. 1462
UPSERT_DATA_FROM_FILE ................................................................. 1464
VACUUM ............................................................................................... 1470
VALIDATE_BACKUP .............................................................................. 1471
VALIDATE_TABLE_BACKUP .................................................................. 1474

System Tables

Introduction ........................................................................................... 1477
SYSALIASES system table .................................................................... 1480
Error Codes (01 to 58)

Introduction........................................................................................................................................... 1518
Class 01 - Warning ................................................................................................................................. 1521
Class 07 - Dynamic SQL Error ................................................................................................................ 1523
Class 08 - Connection Exception ............................................................................................................ 1524
Class 0A - Feature not supported ......................................................................................................... 1527
Class 0P - Invalid role specification ...................................................................................................... 1528
Class 21 - Cardinality Violations ........................................................................................................... 1529
Class 22 - Data Exception ....................................................................................................................... 1530
Class 23 - Constraint Violation .............................................................................................................. 1533
Class 24 - Invalid Cursor State .............................................................................................................. 1534
Class 25 - Invalid Transaction State ..................................................................................................... 1535
Class 28 - Invalid Authorization Specification ..................................................................................... 1536
Class 2D - Invalid Transaction Termination .......................................................................................... 1537
Class 38 - External Function Exception ................................................................. 1538
Class 39 - External Routine Invocation Exception ............................................. 1539
Class 3B - Invalid SAVEPOINT ........................................................................ 1540
Class 40 - Transaction Rollback ....................................................................... 1541
Class 42 - Syntax Error or Access Rule Violation ............................................. 1542
Class 57 - DRDA Network Protocol Execution Failure .................................. 1560
Class 58 - DRDA Network Protocol Protocol Error ........................................ 1561

Error Codes (XS to X0)

Class XBCA - CacheService ........................................................................... 1563
Class XBCM - ClassManager ......................................................................... 1564
Class XBCX - Cryptography .......................................................................... 1565
Class XBM - Monitor ..................................................................................... 1566
Class XCL - Execution exceptions .................................................................. 1567
Class XCW - Upgrade unsupported ............................................................... 1569
Class XCX - Internal Utility Errors ................................................................. 1573
Class XCY - Splice Property Exceptions ....................................................... 1574
Class XCZ - com.splicemachine.db.database.UserUtility ................................ 1575
Class XD00 - Dependency Manager ............................................................... 1576
Class XIE - Import/Export Exceptions ............................................................ 1577
Class XJ - Connectivity Errors ...................................................................... 1579
Class XK - Security Exceptions ..................................................................... 1585
Class XN - Network Client Exceptions .......................................................... 1586
Class XRE - Replication Exceptions ................................................................ 1587

Error Codes (XS to X0)

Class XSAI - Store access.protocol.interface .................................................. 1589
Class XSAM - Store AccessManager ............................................................... 1590
Class XSAS - Store Sort ................................................................................ 1591
Class XSAX - Store access.protocol.XA statement ....................................... 1592
Class XSCB - Store BTree .............................................................................. 1593
Class XSCG0 - Conglomerate ....................................................................... 1594
Class XSCH - Heap ...................................................................................... 1595
Class XSDA - RawStore Data.Generic statement .......................................... 1596
Class XSDB - RawStore Data.Generic transaction ........................................ 1598
Class XSDF - RawStore Data.Filesystem statement ....................................... 1599
Class XSDG - RawStore Data.Filesystem database ....................................... 1600
Class XSLA - RawStore Log.Generic database exceptions ................................ 1601
Class XSLB - RawStore Log.Generic statement exceptions .......................... 1603
Class XSRS - RawStore protocol.Interface statement .................................... 1604
Welcome to Splice Machine!

Welcome to Splice Machine, the database platform for adaptive applications that manage operational processes. This site contains documentation to how you use our products. Visit our company web site to learn more about how our platform works.

* NEW! *

The Splice Machine ML Manager adds machine learning workflow management capabilities to the Splice Machine data platform, giving data scientists an integrated platform for rapid experimentation and robust model management.

To enroll in the exclusive ML Manager beta program, contact Splice Machine at mlmanager@splicemachine.com.

You'll find documentation for the ML Manager here, in this documentation site.

Documentation Versions

This web contains the current customer documentation for version 2.7 of Splice Machine.

If you're using an earlier version, you can find the documentation in these locations:

» Splice Machine Version 2.5 Documentation

Splice Machine Editions

Splice Machine’s database platform is available in several editions, all of which are documented in this web. Click an Edition name in the table below to navigate to the documentation home page for that edition.

<table>
<thead>
<tr>
<th>Splice Machine Edition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Edition</td>
<td>You can configure and deploy our Managed Database Service in the Cloud Edition in a matter of minutes: simply register, adjust a few configuration options for your anticipated data size, cluster power, and backup frequency, and you're up and running within 10-15 minutes.</td>
</tr>
<tr>
<td>Enterprise Edition</td>
<td>You can deploy the Enterprise Edition of our On-Premise Database on a cluster that is managed by Cloudera, MapR, or Hortonworks. This edition is a superset of our Community Edition, adding features such as backup and encryption that are required for enterprise users.</td>
</tr>
</tbody>
</table>
You can deploy the free, open source Cluster Community Edition of our On-Premise Database on a cluster that is managed by Cloudera, MapR, or Hortonworks.

You can install and run the free Standalone Community Edition of our On-Premise Database on a computer running MacOS, Linux, or CentOS. This edition is a great way to quickly learn about and play with the many features of our database platform.

### Feature Comparison
This table summarizes the features that are available in each edition of Splice Machine:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale-Out Architecture</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANSI SQL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Concurrent Acid Transactions</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OLAP and OLTP Resource Isolation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Distributed In-Memory Joins, Aggregations, Scans, and Groupings</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cost-Based Statistics / Query Optimizer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hybrid Row-based and Columnar Storage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Compaction Optimization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Stored Procedures, Triggers, User-Defined Functions</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Apache Kafka-enabled Streaming</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Virtual Table Interfaces</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PL/SQL Support</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backup and Restore Capabilities</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Column Level Access Control</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encryption</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Features, including Kerberos</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDAP Support</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Our Editions page includes additional information about each Splice Machine product edition.

#### Finding Information

This documentation web include three main sections, each of which is accessed through the main menu at the top of each page:

- The **Splice Machine** menu links to topics that apply to all of our database editions, including our Developer topics, Developer tutorials, and SQL Reference Manual.

- The **DB-Service Only** menu links to topics that apply only to our Managed Database Service in the Cloud edition, including our Cloud Manager Guide and guides to using Zeppelin with Splice Machine.

- The **On-Premise-Only** menu links to topics that apply only to the On-Premise editions of our database platform, including our Installation and Administrator’s Guides.

Each topic page in this web also includes a sidebar that links to topics related to the one you are currently viewing. You can hide or show the sidebar by clicking the **Nav** button in the menu at the top of screen.

See the Using our Documentation page to learn more about the content available in our documentation and how to easily navigate among our sections and topics.
Getting Started with Splice Machine

This topic helps you to quickly become productive with your new Splice Machine database by introducing you to some of the major features you’ll be using.

The information on this page assumes that you already have Splice Machine installed; if not, please follow the installation instructions for your platform.

To get started with Splice Machine, we recommend following these simple steps:

1. **Use our command line interpreter** (splice>) to talk directly to your database.
2. **Familiarize yourself with Splice SQL**
3. **Investigate our Developer Topics and Tutorials**
4. **Explore our Developer Tutorials**
5. **Learn About Product-Specific Features**
6. **Optimize Your Use of This Documentation**

1. **Use our Command Line Interpreter**

You can invoke the splice> command line interpreter in your terminal window by navigating to the splicemachine directory on your machine and issuing this command:

```
./sqlshell.sh
```

If you’re connected to a running instance of Splice Machine, you’ll see something like this (please ignore any WARNING messages):
Running Splice Machine SQL shell
For help: "splice> help;"
[INFO] Scanning for projects...

[WARNING]
[WARNING] Some problems were encountered while building the effective model for com.splice
cemachine:splice_machine:jar:2.5.0.1823-SNAPSHOT
[WARNING] 'dependencies.dependency.(groupId:artifactId:type:classifier)' must be uniqu
e: commons-codec:commons-codec:jar -> duplicate declaration of version 1.10 @ com.splice
machine:splice_machine:[unknown-version], /Users/garyh/git/Splice/spliceengine/splice_ma
chine/pom.xml, line 146, column 21
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stabi
lity of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such m
alformed projects.
[WARNING]
[INFO]
[INFO] ------------------------------------------------------------------------------------------------
[INFO] Building splice_machine 2.5.0.1823-SNAPSHOT
[INFO] ------------------------------------------------------------------------------------------------
[INFO]
[INFO] --- exec-maven-plugin:1.4.0:java (default-cli) @ splice_machine ---
SPLICE* - jdbc:splice://localhost:1527/splicedb
* = current connection

Run a SHOW Command
As a first command, try running a simple SHOW TABLES command; since you've not yet created any of your own tables, we'll display the tables defined in the SYS (system) schema:
```sql
splice> SHOW TABLES in SYS;
TABLE_SCHEMA | TABLE_NAME  | CONGLOM_ID | REMARKS
-----------------|-------------|------------|----------
SYS             | SYSALIASES  | 352        |          
SYS             | SYSBACKUP   | 928        |          
SYS             | SYSBACKUPITEMS | 1168   |          
SYS             | SYSCHECKS   | 384        |          
SYS             | SYSCOLPERMS | 640        |          
SYS             | SYSCOLUMNS  | 80         |          
SYS             | SYSCOLUMNSTATS | 1232 |          
SYS             | SYSCONGLOMERATES | 48    |          
SYS             | SYSCONSTRAINTS | 288    |          
SYS             | SYSDEPENDS  | 256        |          
SYS             | SYSFILES    | 240        |          
SYS             | SYSFOREIGNKEYS | 368    |          
SYS             | SYSKEYS     | 272        |          
SYS             | SYSPERMS    | 880        |          
SYS             | SYSphysicalstats | 1248 |          
SYS             | SYSPRIMARYKEYS | 320    |          
SYS             | SYSROLES    | 800        |          
SYS             | SYSROUTINEPERMS | 720    |          
SYS             | SYSSCHEMAPERMS | 1328  |          
SYS             | SYSSCHEMAs  | 32         |          
SYS             | SYSSequences | 864     |          
SYS             | SYSSNAPSHOTS | 1360   |          
SYS             | SYSSSTATEMENTS | 336    |          
SYS             | SYSTABLEPERMS | 624    |          
SYS             | SYSTABLES   | 64         |          
SYS             | SYSTABLESTATS | 1264 |          
SYS             | SYSTOKENS   | 896        |          
SYS             | SYSTRIGGERS | 1408       |          
SYS             | SYSUSERS    | 528        |          
SYS             | SYSVIEWS    | 304        |          
32 rows selected
```

Run a DDL Command

Now let’s run a DDL (Data Definition Language) command to create a table:

```sql
splice> CREATE TABLE myFirstTbl (id SMALLINT, name VARCHAR(32), location VARCHAR(32));
0 rows inserted/updated/deleted
```

This table gets created in the default schema, SPLICE. We can use the SHOW command to verify that our table has been created:
splice> SHOW TABLES IN SPLICE;
TABLE_SCHEM | TABLE_NAME                                        | CONGLOM_ID | REMARKS
KS
----------------------------------------------------------------------------------------
--
SPLICE   | MYFIRSTTBL                                        | 1616       |
1 row selected

Run a DML Command
Let's run a DML (Data Manipulation Language) command to insert some data into the table:

splice> INSERT INTO myFirstTbl VALUES ( 1, 'John Doe', 'San Francisco, CA, USA');
0 rows inserted/updated/deleted

And then examine our table with a SELECT statement:

splice> SELECT * FROM myFirstTbl;
ID    | NAME                            | LOCATION
------------------------------------------------------------------------
1     | John Doe                        | San Francisco, CA, USA USA
1 row selected

Key Things to Know About the Command Line Interpreter
A few key things to remember when using the command line:

- Our Command Line Reference includes complete information about available commands and syntax.
- Installing the rlWrap command line wrapper adds editing, history, and autocompletion to splice> and is highly recommended. See RIWrap Commands Synopsis for more information.
- In general, commands are case-insensitive.
- You must terminate each command with a semicolon (;) character.
- 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;

2. Familiarize Yourself with Splice SQL
Our SQL Reference Manual specifies all of the SQL commands available for use with splice> and via programmatic connections. The manual is organized into syntactic categories, including:
Of particular interest is the *Built-in System Procedures and Functions* section, which describes procedures and functions developed by Splice Machine that add significant functionality to your database. There are a large number of these procedures, which you can use for a wide variety of purposes. You invoke these built-in system procedures with the `CALL` statement. For example, calling this procedure performs an incremental backup:

```
splice> CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE( 'hdfs:///home/backup', 'incremental' );
Statement executed.
```

### 3. Investigate our Developer Topics

Our [Developer Topics and Tutorials](#) and Best Practices Guides contain topics that are useful to database developers, most of which involve using basic SQL functionality and/or built-in system procedures. These topics are organized into multiple sections, including these:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fundamentals</strong></td>
<td>Contains a number of topics that show you how to use fundamental development tools and techniques with Splice Machine, including topics such as running transactions, using triggers, foreign keys, virtual tables, and external tables.</td>
</tr>
<tr>
<td><strong>Functions and Stored Procedures</strong></td>
<td>Describes how to create, modify, and store functions and stored procedures.</td>
</tr>
<tr>
<td><strong>Tuning and Debugging</strong></td>
<td>Helps you to optimize and debug your database queries.</td>
</tr>
<tr>
<td><strong>Accessing Data in the Cloud</strong></td>
<td>Shows you how to access and upload data on Amazon AWS S3.</td>
</tr>
<tr>
<td><strong>Connecting with JDBC</strong></td>
<td>Provides examples of using various programming languages to connect with your database, including Java, Ruby, Python, and Scala.</td>
</tr>
<tr>
<td><strong>Connecting with ODBC</strong></td>
<td>Walks you through installing and using our ODBC driver.</td>
</tr>
</tbody>
</table>
4. Explore our Developer Tutorials and Best Practices

Our Developer Tutorials and Best Practices Guides include deep dives into more complex developer topics, such as:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Securing Your Database</td>
<td>Includes sections on securing your JDBC connections, authentication methods, and authorization of users and roles.</td>
</tr>
<tr>
<td>Importing Data</td>
<td>Compares and walks you through the different ways to import data into your database, and includes detailed examples.</td>
</tr>
<tr>
<td>Using the Database Console</td>
<td>Introduces the <em>Splice Machine Database Console,</em> which is a browser-based tool that you can use to monitor database queries on your cluster in real time.</td>
</tr>
<tr>
<td>Connecting via JDBC and ODBC</td>
<td>Include examples in different programming languages that show you how to programmatically connect to your database.</td>
</tr>
<tr>
<td>Connecting BI Tools</td>
<td>Walks you through connecting specific <em>Business Intelligence</em> tools to your database.</td>
</tr>
</tbody>
</table>

5. Learn About Product-Specific Features

If you're using our On-Premise product, our On-Premise Database Guide contains product-specific sections, including Requirements, Installation, and Administration.

If you're using our Database-as-a-Service product, you'll find our Database-as-Service Guide invaluable for finding your way around the user interface and learning about using the integrated Apache Zeppelin notebooks interface.

6. Optimize Your Use of This Documentation

If you need any help with getting the most out of this documentation system, please see the Using our Documentation topic.
Release Notes for Splice Machine

Welcome to the 2.7 Release of Splice Machine, originally released April, 2018.

Our release notes are presented in these topics:

» This topic contains release notes for the Splice Machine database, which is part of all Splice Machine products.
» The Known Issues and Workarounds topic describes each known issue in the current release and its workaround.
» The Database-as-Service Product Release Notes topic contains release information specific to our Database-as-Service product.
» The On-Premise Product Release Notes topic contains release information specific to our On-Premise product.
» If you’re looking for older release notes, you’ll find them in the Archived Release Notes section, under the Previous Release Docs heading.

Database Release Notes

» Patch Release 2.7.0.1931
» Patch Release 2.7.0.1930
» Patch Release 2.7.0.1929
» Patch Release 2.7.0.1928
» Patch Release 2.7.0.1927
» Patch Release 2.7.0.1926
» Patch Release 2.7.0.1925
» Patch Release 2.7.0.1924
» Patch Release 2.7.0.1923
» Patch Release 2.7.0.1922
» Patch Release 2.7.0.1921
» Patch Release 2.7.0.1920
» Patch Release 2.7.0.1919
» Patch Release 2.7.0.1918
» Patch Release 2.7.0.1917
» Patch Release 2.7.0.1916
» Patch Release 2.7.0.1915
» Patch Release 2.7.0.1914
» Patch Release 2.7.0.1913
Patch Release 2.7.0.1912
Patch Release 2.7.0.1911
Patch Release 2.7.0.1910
Patch Release 2.7.0.1909
Patch Release 2.7.0.1908
Patch Release 2.7.0.1907
Patch Release 2.7.0.1906
Patch Release 2.7.0.1905
Patch Release 2.7.0.1904
Patch Release 2.7.0.1903
Patch Release 2.7.0.1902
Patch Release 2.7.0.1901
Patch Release 2.7.0.1900

The product is available to build from open source (see https://github.com/splicemachine/spliceengine), as well as prebuilt packages for use on a cluster or cloud.
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
» ToDate Function Problem With DD Designator
» 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
» Importing Data with SYCS_IMPORT_DATA from Amazon S3
» HDP 2.6.4 and 2.6.3 OLAP Memory Setting

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.

**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`.

**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 Limitations**
You can only use `ALTER TABLE` to add or remove `PRIMARY KEY` columns if the table being altered is empty.

**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.

**Importing Data with SYSCS_IMPORTED_DATA from Amazon S3**
There are currently two platform-dependent issue for importing data using the SYCS_UTIL.IMPORT_DATA system procedure from Amazon S3 in this release. You only need to pay attention to this note if you're using HDP 2.5.5 or MapR 5.2.0:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDP 2.5.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add a custom setting to your hdfs-site.xml file. Using Ambari, click:</td>
</tr>
<tr>
<td></td>
<td>HDFS-&gt;Configs-&gt;Advanced-&gt;Customer hdfs-site</td>
</tr>
<tr>
<td></td>
<td>and add this setting:</td>
</tr>
<tr>
<td></td>
<td>fs.s3a.impl=com.splicemachine.fs.s3.PrestoS3FileSystem</td>
</tr>
</tbody>
</table>
Platform | Settings
---|---
MapR 5.2.0 | 1. Add the following property setting in `/opt/mapr/hadoop/hadoop-2.7.0/etc/hadoop/hdfs-site.xml` on each node:

```xml
<property>
    <name>fs.s3a.impl</name>
    <value>com.splicemachine.fs.s3.PrestoS3FileSystem</value>
</property>
```

2. Add the following option in `/opt/mapr/hbase/hbase-1.1.1/conf/hbase-env.sh` on each node:

```
SPLICE_HBASE_REGIONSERVER_OPTS="$SPLICE_HBASE_REGIONSERVER_OPTS -Dorg.xerial.snappy.tempdir=/tmp"
```

**NOTE:** These changes may impact non-SpliceMachine services or applications that rely on an implementation other than the Presto implementation; for example, `org.apache.hadoop.fs.s3a.S3AFileSystem`.

### HDP 2.6.4 and 2.6.3 OLAP Memory Setting

If you're using Splice Machine on HDP 2.6.4 or HDP 2.6.3, you need to correct a property setting in the HBase Service Advanced Configuration Snippet (Safety Valve) settings for `hbase-site.xml`.

Replace the . between `olap` and `server` in this property name:

```
splice.olap.server.memory=8192
```

with the _ character to produce the correct property name:

```
splice.olap_server.memory=8192
```

Note that the . character needs to be changed to the _ character in `splice.olap`
Release Notes for Patch Release 2.7.0.1931 (07-Oct-19)

This is the 2.7.0.1931 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Sep 30, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>OlapServer dies with InvalidToken exception</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see https://splice.atlassian.net.
Release Notes for Patch Release 2.7.0.1930 (30-Sep-19)

This is the 2.7.0.1930 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Sep 09, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>There are no updates in this release.</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net).
Release Notes for Patch Release 2.7.0.1929 (09-Sep-19)

This is the 2.7.0.1929 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Sep 03, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Unable to run MERGE_DATA_FROM_FILE() on data that has empty line at the end</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Comment out maven option to listen on port 4002 in platform_it</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net).
Release Notes for Patch Release 2.7.0.1928 (03-Sep-19)

This is the 2.7.0.1928 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on August 26, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>There are no updates in this release.</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see https://splice.atlassian.net.
Release Notes for Patch Release 2.7.0.1927 (26-Aug-19)

This is the 2.7.0.1927 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on August 19, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>There are no updates in this release.</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net).
Release Notes for Patch Release 2.7.0.1926 (19-Aug-19)

This is the 2.7.0.1926 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on August 12, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>SHOW_CREATE_TABLE failed in generating DDL</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Sentry authorizor factory is not recognized</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Restore creates a table with proper content but 0 row count</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>NestedLoopJoin has a lot of overhead even when cloning operation trees</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>inlist predicate push-down to union-all view with different char column size can not handle space trimming well</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see https://splice.atlassian.net.
Release Notes for Patch Release 2.7.0.1925 (12-Aug-19)

This is the 2.7.0.1925 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on August 5, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Too many requests to splice:16 from spark task</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Shutting down HBase / Splice Machine hits Graceful Shutdown Timeout value (10 minutes per our Docs)</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Sentry library is not packaged in 2.7 versions</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Log4j layout to filter out sensitive messages from log</td>
</tr>
<tr>
<td>Story</td>
<td>Internal</td>
<td>Schema Restrict needs to be extended to work when Ranger is used</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1924 (05-Aug-19)

This is the 2.7.0.1924 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on July 22, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>merge_data got stuck due to timeout</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Sentry library is not packaged in 2.7 versions</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>TPC-H Q22 performance regression</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Increase parallelism of Spark table scans for small tables</td>
</tr>
<tr>
<td>Story</td>
<td>Internal</td>
<td>Backup/Restore Statistics for Schema and Table-level Operations</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see https://splice.atlassian.net
Release Notes for Patch Release 2.7.0.1923 (22-Jul-19)

This is the 2.7.0.1923 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on July 08, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Import Date format yyyyMMdd does not work</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Make spark.locality.wait=0</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Make sure backup is complete</td>
</tr>
</tbody>
</table>

For a full list of JIRA’s for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1922 (08-Jul-19)

This is the 2.7.0.1922 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on July 01, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Spark adapter cannot authenticate user name with '.'</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Full table restore creates table and indexes</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1921 (O1-Jul-19)

This is the 2.7.0.1921 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on June 24, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Predicate with constant expression is not pushed down to union view</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1920 (24-Jun-19)

This is the 2.7.0.1920 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on June 10, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Bulk load fails from spark adapter when connecting user is not splice</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Broadcast join or sortmerge join cannot be applied for certain scenarios</td>
</tr>
<tr>
<td>Story</td>
<td>SPLICE-2342</td>
<td>Support builtin function DIGITS used by MicroStrategy</td>
</tr>
<tr>
<td>Story</td>
<td>SPLICE-2345</td>
<td>Support builtin scalar function STRIP used by MicroStrategy</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Investigate HBase Regions in Transition and Delete issues on-site at Customer</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see https://splice.atlassian.net
Release Notes for Patch Release 2.7.0.1919 (10-Jun-19)

This is the 2.7.0.1919 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on May 27, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>We can see the audit logs of all the users in ranger except for the splice user.</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Customer: allow union-all views to benefit from predicate pushdown</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Spark queries have up to 1s extra latency (avg 0.5)</td>
</tr>
<tr>
<td>Story</td>
<td>SPLICE-2341</td>
<td>Builtin Function chr used by Microstrategy</td>
</tr>
<tr>
<td>Story</td>
<td>SPLICE-2343</td>
<td>Support builtin scalar function LEFT used by MicroStrategy</td>
</tr>
<tr>
<td>Story</td>
<td>SPLICE-2344</td>
<td>Support builtin scalar function RIGHT used by MicroStrategy</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1918 (27-May-19)

This is the 2.7.0.1918 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on May 20, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>There are no updates in this release.</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1917 (20-May-19)

This is the 2.7.0.1917 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on May 13, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Splicemachinecontext can't start with kerberos and ldap and kms keystore using user other than 'splice'</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>MERGE_DATA_FROM_FILE fails to update a column with previous null value</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Splice admin user configurable with kerberos authentication</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Customer: Better cut-points when splitting work (scans) for Spark</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see https://splice.atlassian.net
Release Notes for Patch Release 2.7.0.1916 (13-May-19)

This is the 2.7.0.1916 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on May 06, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Feature</td>
<td>SPLICE-579</td>
<td>Killing a Query Through ODBC/JDBC - Support JDBC Cancel() operation</td>
</tr>
<tr>
<td>Improvement</td>
<td>SPLICE-2241</td>
<td>Javadoc and Scaladoc for Spliceengine source and SparkAdapter APIs</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1915 (06-May-19)

This is the 2.7.0.1915 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Apr. 29, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Splicemachinecontext can't start with kerberos and ldap and kms keystore using user other than 'splice'</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Customer: NestedLoopJoin has a lot of overhead even when dealing with few rows</td>
</tr>
</tbody>
</table>

For a full list of JIRA’s for the Community/Open Source software, see https://splice.atlassian.net
Release Notes for Patch Release 2.7.0.1914 (29-Apr-19)

This is the 2.7.0.1914 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Apr. 22, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Data missing in backup</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>NPE exception in the presence of nested loop join and derived table</td>
</tr>
<tr>
<td>Bug</td>
<td>SPLICE-1399</td>
<td>Typo in error message</td>
</tr>
</tbody>
</table>

For a full list of JIRA’s for the Community/Open Source software, see https://splice.atlassian.net
Release Notes for Patch Release 2.7.0.1913 (22-Apr-19)

This is the 2.7.0.1913 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Apr. 15, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>TRIM() function got error when table/column name is same as SPLIT code</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Spark adapter token renewal interfere with Spark-2.3-Cloudera4</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Update SpliceMachine installation guide for HDP to prevent Ambari issue (customer) for other customers</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1912 (15-Apr-19)

This is the 2.7.0.1912 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Apr. 01, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>SPlicemachine SSL is not working when running procedures</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Concurrent readonly query on the same external table failed with 'java.io.FileNotFoundException'</td>
</tr>
<tr>
<td>Sub-task</td>
<td>Internal</td>
<td>Customer: Truncate error for trigger ABS2.DTPE_U06 on the table ABS2.TPERSON</td>
</tr>
<tr>
<td>Sub-task</td>
<td>Internal</td>
<td>Analyze query performance degradation in parallel mode</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Deduplicate Mapr assembly files</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1911 (01-Apr-19)

This is the 2.7.0.1911 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Mar. 25, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Cluster cannot start after force kill with 10,000 tables</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Issues in costing from customer queries</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1910 (25-Mar-19)

This is the 2.7.0.1910 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Mar. 18, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>There are no updates in this release.</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see https://splice.atlassian.net
Release Notes for Patch Release 2.7.0.1909 (18-Mar-19)

This is the 2.7.0.1909 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Mar. 11, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Cannot run count(*) with Ranger enabled</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Avg(distinct) and avg() return different results on non-duplicate column</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Bulk import: error message to report duplicate PK is not user-friendly</td>
</tr>
<tr>
<td>Improvement</td>
<td>SPLICE-2285</td>
<td>Basic predicate simplification</td>
</tr>
<tr>
<td>New Feature</td>
<td>Internal</td>
<td>Add rpms for mapr 6.1.0 to assembly/pom.xml and Jenkins</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Flatten correlated not exists subquery to anti-join instead of left join</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Use average rows per value as estimation for qualified row for equality condition of type &quot;col=expr&quot;</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
**Release Notes for Patch Release 2.7.0.1908 (11-Mar-19)**

This is the 2.7.0.1908 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Mar. 04, 2019.

**Updates in This Release**

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Restore Schema (SYCS_RESTORE_SCHEMA) Not Working</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>HDP Installer setting wrong configuration parameters</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Remove test classes from assemblies</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Improve batched inserts performance</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Remove tables asynchronous for Vacuum</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1907 (04-Mar-19)

This is the 2.7.0.1907 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Feb. 27, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Investigate streaming app ContextManager leak</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1906 (27-Feb-19)

This is the 2.7.0.1906 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Feb. 25, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Wrong spark.version in pom file of profile cdh5.12.2-2.3</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see https://splice.atlassian.net
Release Notes for Patch Release 2.7.0.1905 (25-Feb-19)

This is the 2.7.0.1905 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Feb. 18, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Exception resub.U got propagated to the user and failed query</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Table/Schema backup excludes external table</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Order constraints in show create table result</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Improve error handling in TimestampClient</td>
</tr>
<tr>
<td>Bug</td>
<td>SPLICE-2306</td>
<td>NPE in query with multiple AND'ed and OR'ed IN lists.</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Customer: STRIP</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Export defaults to gzip compression, but import parallelizes on bzip2</td>
</tr>
<tr>
<td>New Feature</td>
<td>Internal</td>
<td>Support &quot;show create table&quot; command to display DDL of a table</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Customer: Add Cloudera 5.12.2 profile with Spark 2.3 rel4</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see https://splice.atlassian.net
Release Notes for Patch Release 2.7.0.1904 (18-Feb-19)

This is the 2.7.0.1904 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Feb. 11, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Deadlock in EmbedPreparedStatement.getMetaData()</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Backup improvements</td>
</tr>
<tr>
<td>Improvement</td>
<td>SPLICE-2291</td>
<td>Improve performance of Base Table lookup with index Access</td>
</tr>
<tr>
<td>New Feature</td>
<td>Internal</td>
<td>Support for Automatically casting columns to correct data types like DB2</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Make RegionServers wait for Master initialization if &quot;java.io.IOException: Table Namespace Manager not fully initialized, try again later&quot; is raised</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Add cdh5.16.x support</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Backup Schema - sys.sysbackups looks incorrect</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1903 (11-Feb-19)

This is the 2.7.0.1903 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Feb. 4, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Schema/table backup copied installed jars accidentally</td>
</tr>
<tr>
<td>Bug</td>
<td>SPLICE-2304</td>
<td>Incorrect results with broadcast join comparing Decimal with Double.</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1902 (4-Feb-19)

This is the 2.7.0.1902 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Jan. 28, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Ambari Splice Machine Service shows a RED Triangle and Splice Machine Install Master as Stopped</td>
</tr>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Restore calculates remaining disk space incorrectly</td>
</tr>
<tr>
<td>Bug</td>
<td>SPLICE-2298</td>
<td>Serialize SQLDecimal type carrying precision and scale value</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Customer: CONCAT alias for</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Fix the spark_avro dependency</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see https://splice.atlassian.net
Release Notes for Patch Release 2.7.0.1901 (28-Jan-19)

This is the 2.7.0.1901 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Jan. 21, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Flush should not block backup</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Deprecate schedule_daily_backup</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
Release Notes for Patch Release 2.7.0.1900 (21-Jan-19)

This is the 2.7.0.1900 Patch Release of Splice Machine, which incorporates updates made since the Patch Release on Jan. 14, 2019.

Updates in This Release

<table>
<thead>
<tr>
<th>Update Type</th>
<th>JIRA-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug</td>
<td>Internal</td>
<td>Multiple lz4 dependencies</td>
</tr>
<tr>
<td>Bug</td>
<td>SPLICE-2274</td>
<td>Splice's ORC reader ignores split offset/length</td>
</tr>
<tr>
<td>Improvement</td>
<td>Internal</td>
<td>Make bulk import sampling rate a database property</td>
</tr>
<tr>
<td>Task</td>
<td>Internal</td>
<td>Update copyright date</td>
</tr>
</tbody>
</table>

For a full list of JIRA's for the Community/Open Source software, see [https://splice.atlassian.net](https://splice.atlassian.net)
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*:

![Splice Machine Documentation](image)

The topbar features these user interface elements:

- Click the *Splice Machine Documentation* title to return you to the home page of our documentation.
- Use the *Top Navigation menus* to navigate to the landing page of each main section of the documentation.
- Use the cascading menus in the *Sidebar Navigation* to navigate to specific topics.
- Click the *Navigation Toggle* to toggle the sidebar navigation off or on.
- Start typing your search terms in the *Search Bar* at the top to find relevant topic.

The sidebar will automatically highlight the location of your current page in the navigation hierarchy whenever you jump to a new page.

**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.

**Documentation Organization**

This site includes documentation for both of our Splice Machine products and two ways to navigate. Most of the topics apply to all of our products; any product-specific topics are gathered into their own top-level (chapter) sections. Here are the main sections of the Splice Machine documentation:
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Welcome</td>
<td>Information about our database: the basis of all Splice Machine products.</td>
</tr>
<tr>
<td>Best Practices Guide</td>
<td>Best practices information for common product usage scenarios.</td>
</tr>
<tr>
<td>Database-as-Service Product</td>
<td>Usage information specific to our cloud-managed database service product.</td>
</tr>
<tr>
<td>Product Info</td>
<td></td>
</tr>
<tr>
<td>Best Practices Guide</td>
<td>Best practices information for common product usage scenarios.</td>
</tr>
<tr>
<td>Database-as-Service Product</td>
<td>Usage information specific to our cloud-managed database service product.</td>
</tr>
<tr>
<td>On-Premise Product Info</td>
<td>Usage information specific to our on-premise database product.</td>
</tr>
<tr>
<td>Reference</td>
<td></td>
</tr>
<tr>
<td>The ML Manager</td>
<td>How to use the Splice Machine Machine Learning Manager for your data science projects.</td>
</tr>
<tr>
<td>SQL Reference Manual</td>
<td>The reference manual for the Splice Machine implementation of SQL.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Table</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Players</td>
<td>The player's name, ID, and other general information.</td>
</tr>
<tr>
<td>Salaries</td>
<td>The salary for each player ID for each season.</td>
</tr>
<tr>
<td>Batting</td>
<td>Batting statistics, per season, for each player ID.</td>
</tr>
<tr>
<td>Fielding</td>
<td>Fielding statistics, per season, for each player ID.</td>
</tr>
<tr>
<td>Pitching</td>
<td>Pitching statistics, per season, for each pitcher's player ID.</td>
</tr>
</tbody>
</table>

The tables were populated with data found on the Internet, primarily from the [baseball-reference.com](http://baseball-reference.com) site.

The remainder of this topic includes this information:

- [Loading the Examples Database](#) walks you through importing the examples database into a new schema in your Splice Machine database.
- [Table Schemas](#) describes the tables.

## Loading the Examples Database

To load the examples database, you need to follow these steps:

1. Click this link to download this small (~12KB) tarball: [https://doc.splicemachine.com/examples/DocExamplesDb.gz](https://doc.splicemachine.com/examples/DocExamplesDb.gz)

2. Unpack the tarball. On MacOS, you can simply double-click the .gz file; on Linux, use the following command:
   ```bash
tar -xvf DocExamplesDb.tar.gz
   ```

3. Navigate to the DocExamplesDb directory that was created when you unpacked the tarball.

4. Use your favorite text editor to edit the CreateDocExamplesDb.sql script file. Scroll to the bottom of the file, where you'll find 5 import statements. You need to update the csv filepath in each of these statements to absolute path of the examples DB directory on your machine.

   For example, one of the import statements looks like this:
If you unpacked the tarball into your Downloads directory, you would change that to the following:

```sql
CALL SYSCS_UTIL.IMPORT_DATA('SPLICEBBALL', 'Players',
    null, '/Users/myName/Downloads/DocExamplesDb/Players.csv',
    null, null, null, null, null, 0, null, true, null);
```

5. In Splice Machine, use the following command line to run the edited script file:

```bash
splice> run '/Users/myName/Downloads/DocExamplesDb/CreateDocExamplesDb.sql';
```

The script file creates the SPLICEBBALL schema and tables, and then imports data from the included csv files into the tables.

### Table Schemas

Our example tables are all stored in a schema named SPLICEBBALL. This section describes the fields in each of our DocsExamplesDb tables.

#### The Players Table

The SPLICEBBALL.Players table contains these columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>SMALLINT</td>
<td>The unique player ID, assigned upon insertion.</td>
</tr>
<tr>
<td>Team</td>
<td>VARCHAR</td>
<td>The abbreviated name of the player’s team.</td>
</tr>
<tr>
<td>DisplayName</td>
<td>VARCHAR</td>
<td>The name we use when displaying this player.</td>
</tr>
<tr>
<td>Position</td>
<td>CHAR(2)</td>
<td>The abbreviation for the player’s main position, e.g. P, C, OF, 1B.</td>
</tr>
<tr>
<td>Birthdate</td>
<td>DATE</td>
<td>The birth date of the player.</td>
</tr>
</tbody>
</table>

#### The Salaries Table

The SPLICEBBALL.Salaries table contains these columns:
### The Batting Table

The `SPLICEBBALL.Batting` contains these columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>SMALLINT</td>
<td>The unique player ID.</td>
</tr>
<tr>
<td>Season</td>
<td>SMALLINT</td>
<td>The season (year).</td>
</tr>
<tr>
<td>Games</td>
<td>SMALLINT</td>
<td>The number of games in which the player batted.</td>
</tr>
<tr>
<td>PlateAppearances</td>
<td>SMALLINT</td>
<td>The number of times the player made a plate appearance.</td>
</tr>
<tr>
<td>AtBats</td>
<td>SMALLINT</td>
<td>The number of official at bats.</td>
</tr>
<tr>
<td>Runs</td>
<td>SMALLINT</td>
<td>The number of runs scores.</td>
</tr>
<tr>
<td>Hits</td>
<td>SMALLINT</td>
<td>How many hits by the player.</td>
</tr>
<tr>
<td>Singles</td>
<td>SMALLINT</td>
<td>How many singles hit by the player.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value is computed by a triggered function.</td>
</tr>
<tr>
<td>Doubles</td>
<td>SMALLINT</td>
<td>How many doubles hit by the player.</td>
</tr>
<tr>
<td>Triples</td>
<td>SMALLINT</td>
<td>How many triples hit by the player.</td>
</tr>
<tr>
<td>HomeRuns</td>
<td>SMALLINT</td>
<td>How many home runs hit by the player.</td>
</tr>
<tr>
<td>RBI</td>
<td>SMALLINT</td>
<td>How many Runs Batted In by the player.</td>
</tr>
<tr>
<td>StolenBases</td>
<td>SMALLINT</td>
<td>How many bases the player stole.</td>
</tr>
<tr>
<td>CaughtStealing</td>
<td>SMALLINT</td>
<td>How many times the player was caught attempting to steal a base.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Walks</td>
<td>SMALLINT</td>
<td>The number of walks the player drew.</td>
</tr>
<tr>
<td>strikeouts</td>
<td>SMALLINT</td>
<td>The number of times the player walked.</td>
</tr>
<tr>
<td>DoublePlays</td>
<td>SMALLINT</td>
<td>How many times the player hit into a double play.</td>
</tr>
<tr>
<td>HitByPitches</td>
<td>SMALLINT</td>
<td>The number of times the player was hit by a pitch.</td>
</tr>
<tr>
<td>SacrificeHits</td>
<td>SMALLINT</td>
<td>The number of sacrifice bunts the player hit.</td>
</tr>
<tr>
<td>SacrificeFlies</td>
<td>SMALLINT</td>
<td>The number of sacrifice flies the player hit.</td>
</tr>
<tr>
<td>IntentionalWalks</td>
<td>SMALLINT</td>
<td>The number of intentional walks issued to the player.</td>
</tr>
<tr>
<td>Average</td>
<td>DECIMAL</td>
<td>The player's batting average.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value is computed by a triggered function.</td>
</tr>
<tr>
<td>TotalBases</td>
<td>SMALLINT</td>
<td>The total number of bases for the player.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value is computed by a triggered function.</td>
</tr>
<tr>
<td>OnBasePercentage</td>
<td>DECIMAL</td>
<td>The percentage of times the player reached base.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value is computed by a triggered function.</td>
</tr>
<tr>
<td>Slugging</td>
<td>DECIMAL</td>
<td>The slugging average of the player.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value is computed by a triggered function.</td>
</tr>
<tr>
<td>OnBasePlusSlugging</td>
<td>DECIMAL</td>
<td>The OPS for the player.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value is computed by a triggered function.</td>
</tr>
</tbody>
</table>

**The Fielding Table**

The SPLICEBBALL.Fielding table contains these columns:
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>SMALLINT</td>
<td>The unique player ID.</td>
</tr>
<tr>
<td>Season</td>
<td>SMALLINT</td>
<td>The season (year).</td>
</tr>
<tr>
<td>FldGames</td>
<td>SMALLINT</td>
<td>How many games the player was in the field for.</td>
</tr>
<tr>
<td>Chances</td>
<td>SMALLINT</td>
<td>The number of fielding chances the player had.</td>
</tr>
<tr>
<td>Putouts</td>
<td>SMALLINT</td>
<td>The number of putouts the player had.</td>
</tr>
<tr>
<td>Assists</td>
<td>SMALLINT</td>
<td>The number of assists the player had.</td>
</tr>
<tr>
<td>Errors</td>
<td>SMALLINT</td>
<td>The number of errors committed by the player.</td>
</tr>
<tr>
<td>FldDoublePlays</td>
<td>SMALLINT</td>
<td>The number of doubles plays in which the player was involved in as a fielder.</td>
</tr>
<tr>
<td>Percentage</td>
<td>DECIMAL</td>
<td>The percentage of opportunities for outs that the player successfully completed.</td>
</tr>
<tr>
<td>TZAboveAverage</td>
<td>SMALLINT</td>
<td>A fielding metric: total zone runs above average for his position.</td>
</tr>
<tr>
<td>TZAboveAveragePer1200</td>
<td>SMALLINT</td>
<td>Total zone runs extrapolated for 1200 innings.</td>
</tr>
<tr>
<td>RunsSaved</td>
<td>SMALLINT</td>
<td>The number of runs saved in the field by the player.</td>
</tr>
<tr>
<td>RunsSavedAboveAvg</td>
<td>SMALLINT</td>
<td>The number of runs saved by the player over the average number saved for his position.</td>
</tr>
<tr>
<td>RangeFactorPerNine</td>
<td>DECIMAL</td>
<td>A fielding metric that evaluates the average number of putouts and assists per nine innings.</td>
</tr>
<tr>
<td>RangeFactorPerGame</td>
<td>DECIMAL</td>
<td>A fielding metric that evaluates the average number of putouts and assists per game played,</td>
</tr>
<tr>
<td>PassedBalls</td>
<td>SMALLINT</td>
<td>The number of passed balls for catchers.</td>
</tr>
<tr>
<td>WildPitches</td>
<td>SMALLINT</td>
<td>The number of wild pitches for pitchers.</td>
</tr>
<tr>
<td>FldStolenBases</td>
<td>SMALLINT</td>
<td>The number of stolen bases given up by a pitcher or catcher.</td>
</tr>
<tr>
<td>FldCaughtStealing</td>
<td>SMALLINT</td>
<td>The number of players caught stealing by a pitcher or catcher.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FldCaughtStealingPercent</td>
<td>DECIMAL</td>
<td>For catchers and pitchers, the percentage of attempted stolen bases that were successful.</td>
</tr>
<tr>
<td>FldLeagueCaughtStealingPercent</td>
<td>DECIMAL</td>
<td>For pitchers and catchers, the league average percentage of attempted stolen bases that were successful.</td>
</tr>
<tr>
<td>Pickoffs</td>
<td>SMALLINT</td>
<td>For pitchers and catchers, the number of runners picked off.</td>
</tr>
<tr>
<td>FldInnings</td>
<td>DECIMAL</td>
<td>The number of innings in which the player was in the field.</td>
</tr>
</tbody>
</table>

**The Pitching Table**

The SPLICEBBALL.Pitching table contains these fields:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>SMALLINT</td>
<td>The unique player ID.</td>
</tr>
<tr>
<td>Season</td>
<td>SMALLINT</td>
<td>The season (year).</td>
</tr>
<tr>
<td>Wins</td>
<td>SMALLINT</td>
<td>How many games the pitcher won.</td>
</tr>
<tr>
<td>Losses</td>
<td>SMALLINT</td>
<td>How many games the pitcher lost.</td>
</tr>
<tr>
<td>Games</td>
<td>SMALLINT</td>
<td>The number of games in which the pitcher appeared.</td>
</tr>
<tr>
<td>GamesStarted</td>
<td>SMALLINT</td>
<td>The number of games the pitcher started.</td>
</tr>
<tr>
<td>GamesFinished</td>
<td>SMALLINT</td>
<td>The number of games the pitcher finished.</td>
</tr>
<tr>
<td>CompleteGames</td>
<td>SMALLINT</td>
<td>The number of complete games by the pitcher.</td>
</tr>
<tr>
<td>Shutouts</td>
<td>SMALLINT</td>
<td>The number of shutout games thrown by the pitcher.</td>
</tr>
<tr>
<td>Saves</td>
<td>SMALLINT</td>
<td>The number of games saved by the pitcher.</td>
</tr>
<tr>
<td>Innings</td>
<td>DECIMAL</td>
<td>The number of innings pitched.</td>
</tr>
<tr>
<td>Hits</td>
<td>SMALLINT</td>
<td>The number of hits given up by the pitcher.</td>
</tr>
<tr>
<td>Runs</td>
<td>SMALLINT</td>
<td>The number of runs give up by the pitcher.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EarnedRuns</td>
<td>SMALLINT</td>
<td>The number of earned runs give up by the pitcher.</td>
</tr>
<tr>
<td>HomeRuns</td>
<td>SMALLINT</td>
<td>How many homeruns the pitcher gave up.</td>
</tr>
<tr>
<td>Walks</td>
<td>SMALLINT</td>
<td>How many walks the pitcher issued.</td>
</tr>
<tr>
<td>IntentionalWalks</td>
<td>SMALLINT</td>
<td>How many intentional walks the pitcher issued.</td>
</tr>
<tr>
<td>Strikeouts</td>
<td>SMALLINT</td>
<td>How many batters the pitchers struck out.</td>
</tr>
<tr>
<td>HitBatters</td>
<td>SMALLINT</td>
<td>How many batters the pitcher hit with a pitch.</td>
</tr>
<tr>
<td>Balks</td>
<td>SMALLINT</td>
<td>How many balks the pitcher committed.</td>
</tr>
<tr>
<td>WildPitches</td>
<td>SMALLINT</td>
<td>How many wild pitches were thrown by the pitcher.</td>
</tr>
<tr>
<td>BattersFaced</td>
<td>SMALLINT</td>
<td>The number of batters faced by the pitcher.</td>
</tr>
<tr>
<td>FieldingIndependent</td>
<td>DECIMAL</td>
<td>A metric (FIP) for pitchers that determines the quality of a pitcher's performance by eliminating plate appearance outcomes that involve defensive play.</td>
</tr>
<tr>
<td>ERA</td>
<td>DECIMAL</td>
<td>The pitcher's earned run average. This value is computed by a triggered function.</td>
</tr>
<tr>
<td>WHIP</td>
<td>DECIMAL</td>
<td>The number of walks and hits per inning pitched by the player. This value is computed by a triggered function.</td>
</tr>
<tr>
<td>HitsPerNine</td>
<td>DECIMAL</td>
<td>The number of hits per nine innings pitched by the player. This value is computed by a triggered function.</td>
</tr>
<tr>
<td>HomeRunsPerNine</td>
<td>DECIMAL</td>
<td>The number of home runs per nine innings pitched by the player. This value is computed by a triggered function.</td>
</tr>
<tr>
<td>WalksPerNine</td>
<td>DECIMAL</td>
<td>The number of walks per nine innings pitched by the player. This value is computed by a triggered function.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>StrikeoutsPerNine</td>
<td>DECIMAL</td>
<td>The number of strikeouts per nine innings pitched by the player. This value is computed by a triggered function.</td>
</tr>
<tr>
<td>StrikeoutsToWalks</td>
<td>DECIMAL</td>
<td>The ratio of strikeouts to walks thrown by the pitcher. This value is computed by a triggered function.</td>
</tr>
</tbody>
</table>
## Trademarks

The following table lists third-party trademark information for products mentioned in this documentation suite:

<table>
<thead>
<tr>
<th>Trademark Holder</th>
<th>Trademarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Web Services, Inc.</td>
<td>AWS and Amazon Elastic Compute Cloud are trademarks of Amazon Web Services, Inc.</td>
</tr>
<tr>
<td>Apache Software Foundation</td>
<td>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.</td>
</tr>
<tr>
<td>Apple Computer, Inc.</td>
<td>Mac OS and OS X are registered trademarks of Apple Computer, Inc.</td>
</tr>
<tr>
<td>Canonical Limited</td>
<td>Ubuntu is a registered trademark of Canonical Limited.</td>
</tr>
<tr>
<td>Cloudera, Inc.</td>
<td>Cloudera is a trademark of Cloudera Corporation.</td>
</tr>
<tr>
<td>DbVis Software AB</td>
<td>DbVisualizer is trademark of DbVis Software AB.</td>
</tr>
<tr>
<td>EasySoft Limited</td>
<td>EasySoft is a trademark of EasySoft Limited.</td>
</tr>
<tr>
<td>Hortonworks, Inc.</td>
<td>Hortonworks and HDP are registered trademarks of Hortonworks, Inc.</td>
</tr>
<tr>
<td>Linus Torvalds</td>
<td>Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.</td>
</tr>
<tr>
<td>MapR Technologies, Inc.</td>
<td>MapR is a registered trademark of MapR Technologies, Inc.</td>
</tr>
<tr>
<td>Microsoft Corporation</td>
<td>Windows is a registered trademark of Microsoft Corporation.</td>
</tr>
<tr>
<td>MongoDB, Inc.</td>
<td>MongoDb is a registered trademark of MongoDB, Inc.</td>
</tr>
<tr>
<td>Oracle Corporation</td>
<td>MySQL is a trademark of Oracle Corporation or its affiliates. Oracle, Java, and JDBC are registered trademarks of Oracle Corporation or its affiliates.</td>
</tr>
<tr>
<td>Pentaho, Inc.</td>
<td>Pentaho is a registered trademark of Pentaho, Inc.</td>
</tr>
<tr>
<td>Red Hat, Inc.</td>
<td>Red Hat, Red Hat Linux, Red Hat Enterprise Linux, and CentOS are registered trademarks of Red Hat, Inc.</td>
</tr>
<tr>
<td>Tableau Software, Inc.</td>
<td>Tableau and Tableau Software are registered trademarks of Tableau Software.</td>
</tr>
<tr>
<td>Trademark Holder</td>
<td>Trademarks</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
</tr>
<tr>
<td>The Open Group</td>
<td>UNIX is a registered trademark of The Open Group in the U.S. and other countries.</td>
</tr>
</tbody>
</table>
# Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
</table>
| **ACID Transactions**       | 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.  
- **Atomicity** means that if one part of the transaction fails, the entire transaction fails.  
- **Consistency** 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.  
- **Isolation** ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed in serial order.  
- **Durability** means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. |
<p>| <strong>Auto-sharding</strong>           | 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. |
| <strong>BI Tools</strong>                | <strong>Business Intelligence Tools</strong>                                                                                                           |
| <strong>CDH</strong>                     | Clouderas <strong>Cloudera Distribution Including Apache Hadoop</strong>, a popular Hadoop platform.                                                    |
| <strong>Column-Oriented Data Model</strong> | 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. |
| <strong>Concurrency</strong>             | The ability for multiple users to access data at the same time.                                                                          |
| <strong>CRM</strong>                     | <strong>Customer Relationship Management</strong>                                                                                                      |
| <strong>Cross-table, cross-row transactions</strong> | A transaction (a group of SQL statements) can modify multiple rows (cross-row) in multiple tables (across tables).                           |
| <strong>CRUD</strong>                    | <strong>Create, Read, Update, Delete.</strong> The four basic functions of persistent storage.                                                            |
| <strong>DAG</strong>                     | <strong>Directed Acyclic Graph.</strong> 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. |
| <strong>Database Statistics</strong>     | A form of dynamic metadata that assists the query optimizer in making better decisions by tracking distribution of values in indexes and/or columns. |</p>
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database Transaction</strong></td>
<td>A sequence of database operations performed as a single logical unit of work.</td>
</tr>
<tr>
<td><strong>ERP</strong></td>
<td><strong>Enterprise Resource Planning</strong> is business management software that a company can use to collect, store, manage and interpret data from many business activities.</td>
</tr>
<tr>
<td><strong>Foreign Key</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Full join support</strong></td>
<td>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 support all five ANSI-standard types of join operations: Inner join, left outer join, right outer join, full outer join, and cross join.</td>
</tr>
<tr>
<td><strong>Hadoop</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>HBase</strong></td>
<td>A column-oriented database management system that is part of the Apache Hadoop framework and runs on top of HDFS.</td>
</tr>
<tr>
<td><strong>HCatalog</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>HDFS</strong></td>
<td><strong>Hadoop Distributed File System</strong>. 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.</td>
</tr>
<tr>
<td><strong>HDP</strong></td>
<td><strong>Hortonworks Data Platform</strong> 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.</td>
</tr>
<tr>
<td><strong>HIVE</strong></td>
<td>Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis.</td>
</tr>
<tr>
<td><strong>HR</strong></td>
<td><strong>Human Resources.</strong></td>
</tr>
<tr>
<td><strong>JDBC</strong></td>
<td><strong>Java Database Connection</strong>. An API specification for connecting with databases using programs written in Java.</td>
</tr>
<tr>
<td><strong>JSON</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>JVM</strong></td>
<td><strong>Java Virtual Machine.</strong> The code execution component of the Java platform.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Key-Value Data Model</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Map Reduce</strong></td>
<td>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:                                                                 1. Each worker node applies the <strong>Map()</strong> function to filter and sort local data. 2. Worker nodes redistribute (shuffle) data based on output keys produced by the <strong>Map()</strong> 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.</td>
</tr>
<tr>
<td><strong>MapR</strong></td>
<td><strong>MapR</strong> 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.</td>
</tr>
<tr>
<td><strong>Multi-partition transactions</strong></td>
<td>A database, like Splice Machine, that allows transactions for a table distributed as multiple partitions across multiple nodes in a cluster.</td>
</tr>
<tr>
<td><strong>MVCC</strong></td>
<td><strong>MultiVersion Concurrency Control</strong> 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.</td>
</tr>
<tr>
<td><strong>MySQL</strong></td>
<td>An open source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL).</td>
</tr>
<tr>
<td><strong>NewSQL</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>NoSQL</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>ODBC</strong></td>
<td><strong>Open DataBase Connectivity</strong>. An open standard API for accessing database management systems, designed to be independent of any specific database or operating system.</td>
</tr>
<tr>
<td><strong>OLAP</strong></td>
<td><strong>OnLine Analytical Processing</strong>. 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.</td>
</tr>
<tr>
<td><strong>OLTP</strong></td>
<td><strong>OnLine Transaction Processing</strong>. A class of information processing systems that facilitate and manage transaction-oriented applications.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Query Optimizer</td>
<td>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.</td>
</tr>
<tr>
<td>Referential Integrity</td>
<td>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.</td>
</tr>
<tr>
<td>Relational Data Model</td>
<td>The model, developed by E.F. Codd, upon which relational database are based. Relational tables have these properties:</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt; Data is presented as a collection of relations</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt; Each relation is depicted as a table</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt; Columns are attributes that belong to the entity modeled by the table</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt; Each row represents a single entity (a record)</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt; Every table has a set of attributes, a key, that unique identifies each entity</td>
</tr>
<tr>
<td>REST</td>
<td>REpresentational State Transfer. 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.</td>
</tr>
<tr>
<td>Rollback</td>
<td>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.</td>
</tr>
<tr>
<td>Scale Out</td>
<td>A database architecture that doesn't rely on a single controller and scales by adding processing power coupled with additional storage.</td>
</tr>
<tr>
<td>Scale Up</td>
<td>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.</td>
</tr>
<tr>
<td>Sharding</td>
<td>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.</td>
</tr>
<tr>
<td>Spark</td>
<td>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.</td>
</tr>
<tr>
<td>Trigger</td>
<td>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.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>YARN</strong></td>
<td><em>Yet Another Resource Negotiator.</em> 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.</td>
</tr>
<tr>
<td><strong>ZooKeeper</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
**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 [Contact Splice Machine Sales] today.

**Feature Comparison**

This table summarizes the features that are available in each edition of Splice Machine:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>On-Demand Compute Nodes and Storage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Managed Backups and Restores</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrated Zeppelin Notebooks</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Splice Machine Cloud Manager</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scale-Out Architecture</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ANSI SQL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Concurrent Acid Transactions</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OLAP and OLTP Resource Isolation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Distributed In-Memory Joins, Aggregations, Scans, and Groupings</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cost-Based Statistics / Query Optimizer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hybrid Row-based and Columnar Storage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Compaction Optimization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Stored Procedures, Triggers, User-Defined Functions</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Apache Kafka-enabled Streaming</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Virtual Table Interfaces</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>PL/SQL Support</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backup and Restore Capabilities</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Column Level Access Control</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
## Features

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Features, including Kerberos</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDAP Support</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Releases and Maintenance Updates</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

---

### Additional Materials and Support

This table summarizes the additional materials and support that are available for each edition of Splice Machine:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tutorials</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Forums</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Videos</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Online Documentation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Community Support</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>24/7 Support via Web and Phone</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Complimentary Account Management Services</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GitHub Repository</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

---

### Licensing

This table compares the pricing policies for the different editions of Splice Machine:

<table>
<thead>
<tr>
<th>Edition</th>
<th>Pricing Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Edition</td>
<td>On compute and storage units per month basis</td>
</tr>
<tr>
<td>Enterprise Edition</td>
<td>On a per node per year basis</td>
</tr>
<tr>
<td>Community Edition</td>
<td>Free</td>
</tr>
</tbody>
</table>
Splice Machine Best Practices Guide

This section contains best practices information for users of Splice Machine, in these subsections:

- **Best Practices: Splice Machine Database**: This section provides best practices and troubleshooting topics that apply to all users of the Splice Machine database:
  - Best Practices: Ingestion guides you to the best available solution for ingesting data when using any Splice Machine product.
  - Best Practices: Native Spark DataSource shows you how to use the Splice Machine Native Spark DataSource for high performance access to your database in Spark.

- **Best Practices: On-Premise Only Topics**: This section contains best practice and troubleshooting information that are specific to our on-premise database product, including:
  - Configuring Security
  - Backing Up Your Data
  - Fine-tuning Performance Options
  - Managing Your Database
  - Summary of Configuration Options
  - Restarting Your Database
  - AfterUpdating Your Splice Machine Software
Best Practices: Ingesting Data - Overview

In this Ingesting Data Best Practices topic, we'll introduce you to the various methods you can use to ingest data into your Splice Machine database, guide you to the best method to use for your specific situation, and then direct you to example(s) that walk you through the steps to implement your ingestion solution.

To get started, use Table 1, below, to determine which use case applies to your situation, then click the How it Works link in the table. You'll see a summary of how to ingest your data, and a link to a separate page that contains one or more detailed examples.

Pick Your Use Case

Where the data that you're ingesting is coming from defines which approach you should take to importing that data into Splice Machine; how to use each approach is described, along with examples, in a section within this topic:

<table>
<thead>
<tr>
<th>Your Use Case</th>
<th>What to Read</th>
<th>Relative Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>You have flat files to import</td>
<td>☛ Ingesting Flat Files</td>
<td>✓✓ to ✓✓✓✓✓ (see below)</td>
</tr>
<tr>
<td>You're writing a Spark app</td>
<td>☛ Ingesting Data in a Spark App</td>
<td>✓✓</td>
</tr>
<tr>
<td>You're writing a streaming app</td>
<td>☛ Ingesting Streaming Data</td>
<td>✓✓✓</td>
</tr>
<tr>
<td>You're accessing data in an external table</td>
<td>☛ Importing Data From an External Table</td>
<td>✓</td>
</tr>
</tbody>
</table>

Importing Flat Files

The most common ingestion scenario is importing flat files into your Splice Machine database; typically, CSV files stored on a local computer, in the cloud, or in HDFS on your cluster. Splice Machine offers two primary methods for importing your flat files: bulk HFile imports and basic flat file imports. Each method has a few variants, which we'll describe later.

Bulk HFile imports are highly performant because the import function pre-splits your data into Hadoop HFiles and imports them directly, which enhances the parallelism of the ingestion processing. When importing larger datasets, this can yield 10x ingestion performance compared to basic import methods. Splice Machine recommends that you use bulk HFile importing large files containing new data.
Basic flat file imports are also performant, and have two important features that may be of importance to you: 1) you can update existing records in addition to adding new records (if and only if the table you’re importing into has a Primary Key), and 2) constraint checking is applied to inserts and updates when using basic import methods.

No matter which method you decide upon, we strongly recommend debugging your ingest process with a small data sample before jumping into importing a large dataset; this will help you to quickly debug any input problems.

Determining Which Method to Use

There are three basic determinants for determining which method to use are:

1. Are you importing all new data or do will you also be updating records in an existing table?
2. Do you need constraint checking applied to the data you’re importing?
3. Are you importing data files larger than 100GB?

The following table summarizes the decision framework for ingesting flat files:

<table>
<thead>
<tr>
<th>All New Data?</th>
<th>Apply Constraints?</th>
<th>Data Size</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>N/A</td>
<td>N/A</td>
<td>Use MERGE_DATA.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>NOTE</strong>: The table into which you’re merging data must have a primary key.</td>
</tr>
<tr>
<td>YES</td>
<td>YES</td>
<td>N/A</td>
<td>Use IMPORT_DATA.</td>
</tr>
<tr>
<td>NO</td>
<td></td>
<td>&lt;=100GB</td>
<td>Use IMPORT_DATA; if that doesn’t yield the performance you need, use BULK_HFILE_IMPORT with automatic splitting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt;100GB</td>
<td>Use BULK_HFILE_IMPORT with automatic splitting; if that doesn’t yield the performance you need, use BULK_HFILE_IMPORT with manual splitting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>NOTE</strong>: Manual splitting adds complexity and requires knowledge of your data: you must determine the key values that will split your input data into approximately evenly-sized partitions (regions).</td>
</tr>
</tbody>
</table>
Note that all of the file import procedures require you to specify the same information about your data, such as: which delimiters are used, how dates and times are formatted, which character set is used, and how to handle invalidly formatted input records. You basically point the procedure at your source data and destination table/index, specify a few parameters, and start loading your data.

**About Bulk HFile Import**

When you use bulk HFile import, Splice Machine first notes the points in your data where it can be split. During ingestion, the `BULK_IMPORT_HFILE` procedure splits the file into temporary Hadoop HFiles (store files) at those points. The store files are then loaded directly into your Region Servers. Ideally, the temporary HFiles are of approximately equal size, which spreads your data evenly across the Region Servers.

Splice Machine offers two bulk import variations:

- **Automatic Splitting** is the easier method because the `BULK_HFILE_IMPORT` procedure takes care of determining the key values that will evenly split your input data into HFiles. The procedure does this by sampling your data to figure out how to split it into (approximately) equal-sized HFiles.

- In **Manual Splitting**, you need to determine those key values yourself, which means that you need to know your data well enough to do that. Because `BULK_HFILE_IMPORT` does not have to sample your data, manual splitting can increase performance, though it adds complexity.

**Contrasting How Standard and Bulk Import Work**

To help you better understand how standard and bulk import work differently, consider these significant differences in import processing:

- When using standard import methods, the data you’re importing is added to your table; when the table fills a region, the table is split across two regions, and the new region starts filling with table data. With bulk import methods, all of the table regions are pre-allocated, and imported data is copied into those regions in parallel.

- When using standard import methods, Splice Machine uses the standard HBase write path, which writes to the HBase Write-ahead-Log (WAL), writes to memstore, flushes to disk, and causes compaction and region splitting. Bulk import bypasses all of this, which results in significantly improved performance.

**For Additional Information and Examples**

- See the [Bulk Importing Flat Files](#) topic in this Best Practices chapter for detailed information about and examples of using bulk HFile ingestion.

- See the [Basic Flat File Ingestion](#) topic in this Best Practices chapter for more information about and examples of using basic flat file ingestion.

**Ingesting Data in a Spark App**

The *Splice Machine Native Spark DataSource* allows you to directly insert data into your database from a Spark DataFrame, which provides great performance by eliminating the need to serialize and deserialize the data. You can write Spark apps (for use with spark-submit) that take advantage of the Native Spark DataSource, or you can use the DataSource in your Zeppelin notebooks.
Ingesting data into your Splice Machine database from Spark is simple: once the data is in a Spark DataFrame, you use the Native Spark DataSource's insert or merge operations to insert the data into a table in your database. These operations are extremely quick, because Splice Machine reads the data into the table directly, without serializing it, sending it over a wire, and deserializing. You can similarly move data from a Splice Machine table into a Spark DataFrame with a single, non-serialized operation.

The *Ingesting Data in a Spark App* topic in this Best Practices chapter contains an example of using the Native Spark DataSource in a standalone Spark application.

### Ingesting Streaming Data

It's also easy to stream data into your Splice Machine database using Spark. The *Ingesting Streaming Data* topic in this Best Practices chapter presents a sample program that uses Spark and Kafka to ingest real-time data into a Splice Machine table.

### Ingesting Data With an External Table

Splice Machine supports *external tables*, which are flat files that you can reference, query, and update inside your database. One common use for external tables is to facilitate ingesting the data in a flat file into a table that you've created in your database. All you need to do is:

- Use the `CREATE EXTERNAL TABLE` statement to specify where your external file is and how it is formatted.
- Use the `INSERT INTO` statement to select data from the external file and insert it into a table in your database.

The *Ingesting Data From an External Table* topic in this Best Practices chapter contains an example of importing data from an ORC file.
Best Practices: Bulk Importing Flat Files

In this topic, we'll walk you through using our highly performant BULK_IMPORT_HFILE procedure to import flat files into your Splice Machine database. Bulk importing splits a data file into temporary HFiles (store files) before it is imported into your database by directly loading the generated StoreFiles into your Region Servers. Ideally, the temporary HFiles are of approximately equal size, which spreads your data evenly across the Region Servers.

NOTE: We recommend using Bulk HFile importing; however, if your input might contain data errors that need checking, you must use our basic import procedures, IMPORT_DATA or MERGE_DATA_FROM_FILE instead, because they perform constraint checking during ingestion.

The remainder of this topic contains these sections:

- About Bulk HFile Import
- Using Automatic Bulk HFile Import
- Using Manual Bulk HFile Import
- Bulk Importing Indexes
- Parameters Used With Bulk Import

You can also use bulk HFiles to speed up performance of the INSERT statement, as shown in the Using Bulk Insert example at the end of this topic.

About Bulk HFile Import

When you use bulk HFile import, Splice Machine first notes the points in your data where it can be split. During ingestion, the BULK_IMPORT_HFILE procedure splits the file into temporary Hadoop HFiles (store files) at those points. The store files are then loaded directly into your Region Servers. Ideally, the temporary HFiles are of approximately equal size, which spreads your data evenly across the Region Servers.

Splice Machine offers two bulk import variations:

- **Automatic Splitting** is the easier method because the BULK_HFILE_IMPORT procedure takes care of determining the key values that will evenly split your input data into HFiles. The procedure does this by sampling your data to figure out how to split it into (approximately) equal-sized HFiles. The next section, automatic bulk hfile ingestion shows you how easily and quickly you can bulk import a file with automatic splitting.

- If you need even better performance from BULK_HFILE_IMPORT and are able to invest some extra effort, you can use Manual Splitting instead of using automatic splitting; this can be particularly valuable for very large datasets. Because BULK_HFILE_IMPORT does not have to sample your data to determine the splits, manual splitting can increase performance, though it adds some complexity and requires that you have enough knowledge of your data to determine where the splits should occur.
If you're going to index the table you're importing, Splice Machine recommends that you create the index prior to using bulk import. This allows the index to also be pre-split into regions, which will prevent downstream bottlenecks.

**Using Automatic Bulk HFile Import**

This section shows you how to use bulk importing of data with automatic sampling, which means that the `BULK_IMPORT_HFILE` procedure samples your data to determine how to evenly (approximately) split it into HFiles. Here's what a call to this procedure looks like, with required parameters highlighted:

```sql
call SYSCS_UTIL.BULK_IMPORT_HFILE('"<schemaName>"', '"<tableName>"', null,
'"<inFilePath>"', null, null, null, null, null, -1,
"<badRecordLogDirectory>"', true, null, '"<temporaryFileDir>"', '"<skipSampling>"');
```

All of the `null` parameter values specify that default values should be used. All of the parameters are described, along with their default values, in Table 1. Here's a call with actual values plugged in:

```sql
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);
```

In the above call, the parameter values have the following meaning:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'TPCH'</td>
<td>Import the data into a table in the <code>TPCH</code> schema in our database.</td>
</tr>
<tr>
<td>'LINEITEM'</td>
<td>Import the data into the <code>LINEITEM</code> table.</td>
</tr>
<tr>
<td>null</td>
<td>Import all columns of data in the file.</td>
</tr>
<tr>
<td>'/TPCH/1/lineitem'</td>
<td>The input file path.</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>null</td>
<td>Character strings in the input file are delimited with <code>&quot;</code> characters.</td>
</tr>
<tr>
<td>null</td>
<td>Timestamp values are in <code>yyyy-MM-dd HH:mm:ss</code> format.</td>
</tr>
<tr>
<td>null</td>
<td>Date values are in <code>yyyy-MM-dd</code> format.</td>
</tr>
<tr>
<td>null</td>
<td>Time values are in <code>HH:mm:ss</code> format.</td>
</tr>
<tr>
<td>-1</td>
<td>Any number of rejected (bad) records will be tolerated.</td>
</tr>
<tr>
<td>'/BAD'</td>
<td>Information about any bad records is logged in this directory.</td>
</tr>
<tr>
<td>true</td>
<td>Each record is contained in one line in the input file.</td>
</tr>
<tr>
<td>null</td>
<td>The input uses UTF-8 character encoding.</td>
</tr>
<tr>
<td>'hdfs:///tmp/test_hfile_import/'</td>
<td>The HDFS directory where temporary HFiles are stored until the import completes.</td>
</tr>
<tr>
<td>false</td>
<td>Use automatic sampling.</td>
</tr>
</tbody>
</table>

**Example: Automatic Bulk HFile Import**

To use Bulk HFile import with automatic splitting, follow these steps:

1. **Create the table in your Splice Machine database:**

```sql
splice> 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_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)
);
```

2. **Create the index in your Splice Machine database:**

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

3. **Create a directory on HDFS for the temporary HFiles. 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.

4. **Import your data:**

```java
call SYSCS_UTIL.BULK_IMPORT_HFILE('TPCH', 'LINEITEM', null,
    's3a://splice-benchmark-data/flat/TPCH/1/lineitem', '|', null, null, null,
    null, -1,
    '/BAD', true, null, 'hdfs:///tmp/test_hfile_import/', false);
```

Note that the final parameter, `skipSampling`, is `false` in the above call; this tells `BULK_IMPORT_HFILE` to split the data based on its own sampling. All of the parameters are summarized in **Table 1** below.

### Using Manual Bulk HFile Import

If you don't get the performance you're hoping from with automatic bulk import, you can manually specify how `BULK_IMPORT_HFILE` should split your data by providing a CSV file with key values in it. You pass the CSV file in to the `SPLIT_TABLE_OR_INDEX` procedure to pre-split the data, and then call the `BULK_IMPORT_HFILE` procedure to import the HFiles, as shown in the example below.

**Pre-splitting** means that `SPLIT_TABLE_OR_INDEX` notes where to split the table that you're importing into during ingestion, and stores that split information. When you call `BULK_IMPORT_HFILE` with the `skipSampling` parameter set to `true`, the procedure makes use of that information to split the table.

The remainder of this section contains these subsections:

- **Manual Bulk Import Basics** provides basic information about manual bulk import.
- **Bulk Importing Indexes** presents options for indexes on bulk imported tables.
- **Example: Manual Bulk HFile Import** shows an example of using bulk import.

### Manual Bulk Import Basics

The basic flow for manual bulk import is:

2. Optionally create an index on that table.
3. Determine key values for the table that will split it evenly and store them in a CSV file.
4. Call `SPLIT_TABLE_OR_INDEX` with the table name and that CSV file.
5. Optionally: a. Determine key values that will split the index evenly and store them in a CSV file. b. Call `SPLIT_TABLE_OR_INDEX` with the table name, index name, and that CSV file.
6. Call `BULK_IMPORT_HFILE` with `skipSampling=true` to import the file.
You can use the `SPLIT_TABLE_OR_INDEX` in two ways:

- To compute splits for a table into which you’re about to bulk import a file: specify a table name, but not an index name, and the name of the CSV file that contains the table split key values.
- To compute splits for an index on a table into which you’re about to bulk import a file: specify a table name, an index name, and the name of the CSV that contains the index split key values.

In both cases, `SPLIT_TABLE_OR_INDEX` associates the split key information with the table so that `BULK_IMPORT_HFILE` can use it during ingestion. If you pre-split your table, we also recommend that you pre-split your index; otherwise, your index table could eventually become a performance bottleneck. If your index includes your primary key, this is simple: you can use the same split key values for both.

When you call `BULK_IMPORT_HFILE` with `skipSampling=true`, it assumes that you have previously called `SPLIT_TABLE_OR_INDEX` to pre-split the table; if you call it without first pre-splitting the table, `BULK_IMPORT_HFILE` will store the entire file in one region; such a table will suffer from very poor performance until you perform a major compaction.

**Example: Manual Bulk HFile Import**

To use Bulk HFile import by manually specifying key values, follow the steps below:

1. **Create the table in your Splice Machine database:**

   ```sql
   splice> 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.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)    
  );
   ```

2. **Create the index in your Splice Machine database:**
splice> CREATE INDEX L_SHIPDATE_IDX on TPCH.LINEITEM(
    L_SHIPDATE,
    L_PARTKEY,
    L_EXTENDEDPRICE,
    L_DISCOUNT
);
7. Specify the column names in that CSV file in the columnList parameter; in our example, the index columns are:

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

8. Invoke SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX to pre-split your index file:

```sql
call SYCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX('TPCH',
'L_URIITEM', '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);
```

9. Call SYCS_UTIL.GET_REGIONS to verify that the table and index are split as expected:

```sql
CALL SYCS_UTIL.GET_REGIONS( 'TPCH', 'LINEITEM', 'L_SHIPDATE_IDX',
null, null, '|', null, null, null, null);
```

10. Import the file into your database:

```sql
call SYCS_UTIL.BULK_IMPORT_HFILE('TPCH', 'LINEITEM', null,
's3a://splice-benchmark-data/flat/TPCH/1/lineitem', '|', null, null, null,
null, null, -1, '/BAD', true, null,
'hdfs:///tmp/test_hfile_import/', true);
```

Note that the final parameter, skipSampling is true in the above call; this tells BULK_IMPORT_HFILE that the data has already been pre-split into HFiles.

### Parameters Used With Bulk Import

The following table summarizes the parameters you use when calling the BULK_IMPORT_HFILE and SYCS_SPLIT_TABLE_OR_INDEX procedures.

---

**NOTE:** The Data Ingestion Parameter Values topic in this chapter provides reference information for all of these parameters.
### Table 1: Bulk HFile Import Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaName</td>
<td>The schema to import into.</td>
</tr>
<tr>
<td>tableName</td>
<td>The table to import into.</td>
</tr>
<tr>
<td>insertColumnList</td>
<td>A list of the columns to import; The default is to import all columns.</td>
</tr>
<tr>
<td>fileOrDirectoryName</td>
<td>The file or directory of files to import.</td>
</tr>
<tr>
<td>columnDelimiter</td>
<td>The character used to separate columns in the input; The default is the comma (,) character.</td>
</tr>
<tr>
<td>characterDelimiter</td>
<td>The character used to delimit strings in the imported data; the default is the double-quote (&quot;) character.</td>
</tr>
<tr>
<td>timestampFormat</td>
<td>The format of timestamps stored in the file; The default is &quot;yyyy-MM-dd HH:mm:ss&quot;.</td>
</tr>
<tr>
<td>dateFormat</td>
<td>The format of date values stored in the file; The default is &quot;yyyy-MM-dd&quot;.</td>
</tr>
<tr>
<td>timeFormat</td>
<td>The format of time values stored in the file; The default is &quot;HH:mm:ss&quot;.</td>
</tr>
<tr>
<td>badRecordsAllowed</td>
<td>The number of rejected (bad) records that are tolerated before the import fails. A value of <code>0</code> means that the import terminates as soon as a single bad record is detected; a value of <code>-1</code> means that all bad records are tolerated and logged.</td>
</tr>
<tr>
<td>badRecordDirectory</td>
<td>The directory in which bad record information is logged.</td>
</tr>
<tr>
<td>oneLineRecords</td>
<td>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.</td>
</tr>
<tr>
<td>charset</td>
<td>The character encoding of the import file. The default value is UTF-8.</td>
</tr>
<tr>
<td>bulkImportDirectory</td>
<td>The HDFS directory where you want the temporary HFiles stored until the import completes.</td>
</tr>
<tr>
<td>skipSampling</td>
<td>Specify <code>true</code> if you've pre-split the data; <code>false</code> to have this procedure determine the splits by sampling the data.</td>
</tr>
</tbody>
</table>

### Using Bulk Insert

There's one more way to use bulk HFiles for ingesting data into your Splice Machine tables: you can add a set of hints to an `INSERT` statement that tell the database to use bulk import technology to insert a set of query results into a table.
Splice Machine allows you to specify optimization hints; one of these hints, bulkImportDirectory, can be used to perform bulk loading with the SQL INSERT statement.

You do this by adding these hints to the INSERT:

- The `bulkImportDirectory` hint is used just as it is with the `BULK_HFILE_IMPORT` procedure: to specify where to store the temporary HFiles used for the bulk import.
- The `useSpark=true` hint tells Splice Machine to use the Spark engine for this insert. This is required for bulk HFile inserts.
- The `skipSampling` hint is used just as it is with the `BULK_HFILE_IMPORT` procedure (see Table 1): to tell the bulk insert to compute the splits automatically or that the splits have been supplied manually.

Here's a simple example:

```sql
DROP TABLE IF EXISTS myUserTbl;
CREATE TABLE myUserTbl AS SELECT
    user_id,
    report_date,
    type,
    region,
    country,
    access,
    birth_year,
    gender,
    product,
    zipcode,
    licenseID
FROM licensedUserInfo
WITH NO DATA;

INSERT INTO myUserTbl --splice-properties bulkImportDirectory='/tmp',
useSpark=true,
skipSampling=false
SELECT * FROM licensedUserInfo;
```

For Additional Information

Our SQL Reference Manual includes reference pages for each of these system procedures, which include full information about the parameters, additional examples, and discussion of handling special cases and input errors:

- SYSCS_UTIL.BULK_IMPORT_HFILE
- SYCS_UTIL.COMPUTE_SPLIT_KEY
- SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX
Best Practices: Basic Flat File Ingestion

This topic shows you how to use Splice Machine’s basic data ingestion methods, IMPORT_DATA and MERGE_DATA_FROM_FILE, to import data from flat files into your database. These highly performant procedures provide numerous data handling options, and perform constraint checking, which means that they detect and report on erroneous records (bad data) in the input file.

If you are ingesting all new data, use IMPORT_DATA; if you are also ingesting updates to existing records in your database table, use MERGE_DATA_FROM_FILE. You can only merge data into a table that has a primary key.

The remainder of this topic contains these sections:

- About Basic Flat File Ingestion
- Example: Basic Import of a Flat File
- Example: Basic Merge of a Flat File
- Parameters Used With the Basic Import Procedures

Our Bulk HFile Import procedure, BULK_HFILE_IMPORT, offers boosted ingestion speed when importing large (> 100GB) data sets, but does not perform constraint checking.

About Basic Flat File Ingestion

Here’s what a call to the IMPORT_DATA procedure looks like, with required parameters highlighted:

```sql
CALL SYSCS_UTIL.IMPORT_DATA(
  '<schemaName>',
  '<tableName>',
  null,
  '<inFilePath>',
  null, null, null, null, null, -1,
  '<badRecordLogDirectory>', true, null);
```

All of the null parameter values specify that default values should be used. All of the parameters are described, along with their default values, in Table 1. Here’s a call with actual values plugged in:

```sql
CALL SYSCS_UTIL.IMPORT_DATA(
  'SPLICE',
  'playerTeams',
  null, 'myData.csv',
  null, null, null, null, null, 0, 'importErrsDir', true, null);
```

**NOTE:** The MERGE_DATA_FROM_FILE procedure uses exactly the same parameters.

In the above calls, the parameter values have the following meaning:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'SPLICE'</td>
<td>Import the data into a table in the <code>SPLICE</code> schema in our database.</td>
</tr>
<tr>
<td>'playerTeams'</td>
<td>Import the data into the <code>playerTeams</code> table.</td>
</tr>
<tr>
<td>null</td>
<td>Import all columns of data in the file.</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>'myData.csv'</td>
<td>The input file path.</td>
</tr>
<tr>
<td>null</td>
<td>Columns in the input file are separated by the , character.</td>
</tr>
<tr>
<td>null</td>
<td>Character strings in the input file are delimited with &quot; characters.</td>
</tr>
<tr>
<td>null</td>
<td>Timestamp values are in yyyy-MM-dd HH:mm:ss format.</td>
</tr>
<tr>
<td>null</td>
<td>Date values are in yyyy-MM-dd format.</td>
</tr>
<tr>
<td>null</td>
<td>Time values are in HH:mm:ss format.</td>
</tr>
<tr>
<td>0</td>
<td>Zero tolerance for bad records: any input error will terminate the import.</td>
</tr>
<tr>
<td>'importErrsDir'</td>
<td>Information about any bad records is logged in this directory.</td>
</tr>
<tr>
<td>true</td>
<td>Each record is contained in one line in the input file.</td>
</tr>
<tr>
<td>null</td>
<td>The input uses UTF-8 character encoding.</td>
</tr>
</tbody>
</table>

**Example: Basic Import of a Flat File**

Here’s a very basic example of importing a flat file into a table in your Splice Machine database. Follow these steps:

1. **Create a simple table named testImport:**

   ```sql
   CREATE SCHEMA test;
   SET SCHEMA test;

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

2. **Access a simple file named ttest.csv from an S3 bucket on AWS. That file contains this data:**

   | 0 | 0 |
   | 1 | 2 |
   | 2 | 4 |
   | 3 | 6 |
   | 4 | 8 |
3. **Use IMPORT_DATA to import that data into the testImport table:**

```sql
splice> CALL SYSCS_UTIL.IMPORT_DATA('TEST', 'testImport', null,
's3a://mypublicbucket/ttest.csv',
'|', null, null, null, null, 0,
'hdfs:///tmp/test_import/', false, null);
```

<table>
<thead>
<tr>
<th>rowsImported</th>
<th>failedRows</th>
<th>files</th>
<th>dataSize</th>
<th>failedLog</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>20</td>
<td>NONE</td>
</tr>
</tbody>
</table>

Note that this IMPORT_DATA call logs bad import records to a file on HDFS, and uses almost all default parameter values. The exception: our data file uses the | to delimit columns. All of the parameters are summarized in Table 1 below.

4. **Use a SELECT statement to verify that all went well:**

```sql
splice> SELECT * FROM testImport;
```

```
+---+---+---+---+
<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
<th>C1</th>
<th>D1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>999</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>999</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>999</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>4</td>
<td>999</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>5</td>
<td>999</td>
</tr>
</tbody>
</table>
+---+---+---+---+
```

**Example: Basic Merge of a Flat File**

Here's a very basic example of using MERGE_DATA_FROM_FILE to add new records and update a few existing records in a table. This example ingests into the same table that we just used in the IMPORT_DATA example above.

1. **Access a simple file named mergetest.csv from an S3 bucket on AWS. That file contains the following data. Note that the rows with key values 2 and 4 already exist in the table:**

   ```
   2 | 22  
   4 | 44  
   5 | 55  
   6 | 66  
   ```

2. **Use MERGE_DATA to import that data into the testImport table:**
3. **Use a SELECT statement to verify that all went well:**

```sql
splice> SELECT * FROM testImport;
A1  | B1  | C1   | D1  
---|-----|------|-----
0   | 0   | 1    | 999 
1   | 2   | 2    | 999 
2   | 22  | 3    | 999 
3   | 6   | 4    | 999 
4   | 44  | 5    | 999 
5   | 55  | 10001| 999 
6   | 66  | 10002| 999 
```

7 rows selected

Note that this `MERGE_DATA_FROM_FILE` call uses exactly the same parameter values as does the previous call to `IMPORT_DATA`, with the exception of importing a different file. As you can see, two rows (A1=2 and A1=4) were updated with new B1 values, and two new rows were added by this merge call.

### Parameters Used With the Basic Import Procedures

The following table summarizes the parameters you use when calling the `IMPORT_DATA` or `MERGE_DATA_FROM_FILE` procedures.

**NOTE:** The Data Ingestion Parameter Values topic in this chapter provides reference information for all of these parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaName</td>
<td>The schema to import into.</td>
</tr>
<tr>
<td><strong>Parameter Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tableName</td>
<td>The table to import into.</td>
</tr>
<tr>
<td>insertColumnList</td>
<td>A list of the columns to import; The default is to import all columns.</td>
</tr>
<tr>
<td>fileOrDirectoryName</td>
<td>The file or directory of files to import.</td>
</tr>
<tr>
<td>columnDelimiter</td>
<td>The character used to separate columns in the input; The default is the comma (,) character.</td>
</tr>
<tr>
<td>characterDelimiter</td>
<td>The character used to delimit strings in the imported data; the default is the double-quote (&quot;), character.</td>
</tr>
<tr>
<td>timestampFormat</td>
<td>The format of timestamps stored in the file; The default is &quot;yyyy-MM-dd HH:mm:ss&quot;.</td>
</tr>
<tr>
<td>dateFormat</td>
<td>The format of date values stored in the file; The default is &quot;yyyy-MM-dd&quot;.</td>
</tr>
<tr>
<td>timeFormat</td>
<td>The format of time values stored in the file; The default is &quot;HH:mm:ss&quot;.</td>
</tr>
<tr>
<td>badRecordsAllowed</td>
<td>The number of rejected (bad) records that are tolerated before the import fails. A value of <code>0</code> means that the import terminates as soon as a single bad record is detected; a value of <code>-1</code> means that all bad records are tolerated and logged.</td>
</tr>
<tr>
<td>badRecordDirectory</td>
<td>The directory in which bad record information is logged.</td>
</tr>
<tr>
<td>oneLineRecords</td>
<td>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.</td>
</tr>
<tr>
<td>charset</td>
<td>The character encoding of the import file. The default value is UTF-8.</td>
</tr>
</tbody>
</table>

### For Additional Information

Our SQL Reference Manual includes reference pages for each of these system procedures, which include full information about the parameters, additional examples, and discussion of handling special cases and input errors:

- SYSCS_UTIL.IMPORT_DATA
- SYSCS_UTIL.MERGE_DATA_FROM_FILE
Best Practices: Ingesting Data in a Spark App

The Splice Machine Native Spark DataSource allows you to directly insert data into your database from a Spark DataFrame, which provides great performance by eliminating the need to serialize and deserialize the data. You can write Spark programs that take advantage of the Native Spark DataSource, or you can use it in your Zeppelin notebooks.

This topic presents two examples of using the Splice Machine Native Spark DataSource to load data with Spark into a table in your database:

- **Using the Native Spark DataSource to Ingest Data in Zeppelin**
- **Using the Native Spark DataSource with spark-submit to Ingest Data**

For an overview of best practices for data ingestion, see Best Practices: Ingesting Data, in this Best Practices chapter.

### Using the Native Spark DataSource to Ingest Data in Zeppelin

This section presents a simple Zeppelin notebook example, written in Scala, of moving data between a Spark DataFrame and a Splice Machine table:

1. **Create the context:**
   
   Using the `%spark` interpreter in Zeppelin, create an instance of the SplicemachineContext class; this class interacts with your Splice Machine cluster in your Spark executors, and provides the methods that you can use to perform operations such as directly inserting into your database from a DataFrame:

   ```scala
   %spark
   import com.splicemachine.spark.splicemachine._
   import com.splicemachine.derby.utils._

   val JDBC_URL = "jdbc:splice://<yourJDBCUrl>:1527/splicedb;user=<yourUserId>;password=<yourPassword>"
   val splicemachineContext = new SplicemachineContext(JDBC_URL)
   ```

2. **Create a DataFrame in Scala and populate it:**
   
   Again using the `%spark` interpreter in Zeppelin, we create a DataFrame and populate it with a little data:

   ```scala
   %spark
   val carsDF = Seq(
   (1, "Toyota", "Camry"),
   (2, "Honda", "Accord"),
   (3, "Subaru", "Impreza"),
   (4, "Chevy", "Volt")
   ).toDF("NUMBER", "MAKE", "MODEL")
   ```

   Though this DataFrame contains only a very small amount of data, the code in this example can be scaled to DataFrames of any size.
3. **Create a Table in your Splice Machine Database:**

Now we'll create a simple table in our database, using the `%splicemachine` interpreter in Zeppelin:

```sql
%splicemachine
create table mySchema.carsTbl ( number int primary key,
                                make  varchar(20),
                                model varchar(20) );
```

4. **Populate the Database Table from the DataFrame:**

To ingest all of the data in the DataFrame into your database, simply use the `Insert` method of the Splice Machine Native Spark DataSource:

```sql
%spark
splicemachineContext.insert( carsDF, "mySchema.carsTbl");
```

Ingesting data in this way is extremely performant because it requires no serialization or deserialization and works with any Spark DataFrame. You can also use the Native Spark DataSource to quickly query your database, and to update or delete records.

5. **Verify that All Went Well:**

```sql
%splicemachine
select * from mySchema.carsTbl;
```

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>MAKE</th>
<th>MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Toyota</td>
<td>Camry</td>
</tr>
<tr>
<td>2</td>
<td>Honda</td>
<td>Accord</td>
</tr>
<tr>
<td>3</td>
<td>Subaru</td>
<td>Impreza</td>
</tr>
<tr>
<td>4</td>
<td>Chevy</td>
<td>Volt</td>
</tr>
</tbody>
</table>

**Using the Native Spark DataSource with spark-submit to Ingest Data**

This section presents a discussion of and sample code for a standalone program submitted with `spark-submit` that uses the Splice Machine Native Spark DataSource to ingest data into a table.

**NOTE:** All of the files required to build and run this program are available here:

`.examples/SparkAppSubmit.tar.gz`

We show you how to create and run this example in these subsections:

1. **The Submit Script** presents the `spark-submit.sh` script and describes the variables that you need to modify before running the script.

2. **The Example Code** section presents the code for our example program that ingests data using the Splice Machine Native Spark DataSource.
3. **Build and Run the Example Program** walks you through building and running the sample code.

### The Submit Script

You can use the supplied `spark-submit.sh` script to run a Spark program. Here's a version of this script:

```bash
#!/bin/bash
export SPARK_KAFKA_VERSION=0.10

TargetTable=LINEITEM
TargetSchema=TPCH
RSHostName=srv132
SpliceConnectPort=1527
UserName=yourDBUserId
UserPassword=yourDBPassword
HdfsHostName=srv131
HdfsPort=8020
CsvFilePath=/TPCH/1/lineitem

spark2-submit \ 
  --conf "spark.dynamicAllocation.enabled=false" \ 
  --conf "spark.streaming.stopGracefullyOnShutdown=true" \ 
  --conf "spark.streaming.concurrentJobs=1" \ 
  --conf "spark.task.maxFailures=2" \ 
  --conf "spark.driver.memory=4g" \ 
  --conf "spark.driver.cores=1" \ 
  --conf "spark.driver.extraJavaOptions=-verbose:class" \ 
  --conf "spark.executor.extraJavaOptions=-verbose:class" \ 
  --conf "spark.executor.extraClassPath=/etc/hadoop/conf/:/etc/hbase/conf/:/opt/cloudera/parcels/SPICEMACHINE/lib/*:/opt/cloudera/parcels/CDH/lib/hbase/lib/*" \ 
  --conf "spark.driver.extraClassPath=/etc/hadoop/conf/:/etc/hbase/conf/:/opt/cloudera/parcels/SPICEMACHINE/lib/*:/opt/cloudera/parcels/SPARK2/lib/spark2/jars/*:/opt/cloudera/parcels/CDH/lib/hbase/lib/*" \ 
  --name "Spark Adapter Ingest" \ 
  --jars "splicemachine-cdh5.12.2-2.2.0.cloudera2_2.11-2.7.0.1908.jar" \ 
  --class com.splicemachine.example.Main \ 
  --master yarn --deploy-mode cluster --num-executors 10 --executor-memory 10G --executor-cores 4 \ 
  target/example-1.0-SNAPSHOT.jar \ 
$TargetTable $TargetSchema $RSHostName $SpliceConnectPort $UserName $UserPassword $HdfsHostName $HdfsPort $CsvFilePath
```

Before submitting your Spark program with this script, you need to modify some of the values (at least the highlighted) at the top of the script; for our sample program, these are the values, which are summarized in the table below:
Table 1: spark-submit script variables

<table>
<thead>
<tr>
<th>Script Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetTable</td>
<td>The name of the table in your Splice Machine database into which you are importing data.</td>
</tr>
<tr>
<td>TargetSchema</td>
<td>The name of the schema in your database to which the table belongs.</td>
</tr>
<tr>
<td>RSHostName</td>
<td>The region server for connecting to your database.</td>
</tr>
<tr>
<td>SpliceConnectPort</td>
<td>The port number for connecting to your database.</td>
</tr>
<tr>
<td>UserName</td>
<td>The user name for connecting to your database.</td>
</tr>
<tr>
<td>UserPassword</td>
<td>The user password for connecting to your database.</td>
</tr>
<tr>
<td>HdfsHostName</td>
<td>The region server for connecting to HDFS.</td>
</tr>
<tr>
<td>HdfsPort</td>
<td>The port number for connecting to HDFS.</td>
</tr>
<tr>
<td>CsvFilePath</td>
<td>The HDFS path to the CSV file you’re importing.</td>
</tr>
</tbody>
</table>

The Example Code

This section presents a simple example of ingesting data from a CSV file into a database table with the Splice Machine Native Spark DataSource.

This code does the following:

1. Configures variables from the parameters in the spark-submit.sh script.
2. Creates a Spark session.
3. Creates a JDBC URL for connecting to your Splice Machine database.
4. Creates a Splice Machine Native Spark DataSource context (splicemachineContext) with that URL.
5. Creates a Spark DataFrame from the CSV file that you’re importing.
6. Inserts the data from the DataFrame into your database.

Here's the code:
package com.splicemachine.example;

import org.apache.hadoop.security.UserGroupInformation;
import org.apache.spark.SparkConf;
import java.net.URL;
import java.net.URLClassLoader;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.Row;
import java.io.IOException;
import com.splicemachine.derby.impl.SpliceSpark;
import com.splicemachine.spark.splicemachine.*;
import org.apache.spark.sql.types.DataTypes;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;
import org.apache.spark.sql.types.Metadata;
import org.apache.spark.sql.Dataset;

public class Main {
    public static void main(String[] args) throws Exception {
        if(args.length < 9) {
            System.err.println("Incorrect number of params ");
            return;
        }

        final String inTargetTable = args[0];
        final String inTargetSchema = args[1];
        final String inHostName = args[2];
        final String inHostPort = args[3];
        final String inUserName = args[4];
        final String inUserPassword = args[5];
        final String inHDFSHostName = args[6];
        final String inHDFSPort = args[7];
        final String inCSVFilePath = args[8];

        SparkConf conf = new SparkConf();
        SparkSession spark = SparkSession.builder().appName("Ingest").config(conf).getOrCreate();
        SpliceSpark.setContext(spark.sparkContext());

        // Construct a connection string
        String dbUrl = "jdbc:splice://" + inHostName + ":" + inHostPort
                        + "/splicedb;user=" + inUserName
                        + ";" + "password=" + inUserPassword;

        // Create a SplicemachineContext based on the provided DB connection
        SplicemachineContext splicemachineContext = new SplicemachineContext(dbUrl);
// Set target table name and schema name
String SPLICE_TABLE_ITEM = inTargetSchema + "." + inTargetTable;

StructType schema = splicemachineContext.getSchema(SPLICE_TABLE_ITEM);

// Create a DataFrame from a specified file
Dataset<Row> df = spark.read().format("com.databricks.spark.csv").option("delimeter","|").schema(schema)
.load("hdfs://" + inHDFSHostName + ":" + inHDFSPort + inCSVFilePat);

splicemachineContext = new SplicemachineContext(dbUrl);
// Import the data
splicemachineContext.insert(df, SPLICE_TABLE_ITEM);

}

---

Build and Run the Example Program

You can download, build, and run this example program as follows:

1. **Click this link to download the example tarball:** ./examples/SparkAppSubmit.tar.gz
2. **Build the program with this command:**
   
   mvn clean install
3. **Copy the lineitem.csv file to HDFS. For example:**
   
   hadoop fs -copyFromLocal lineitem.csv /TPCH/1/lineitem
4. **Modify the spark-submit.sh script for your environment. The values you may need to modify are at the top of the script:**
   To run this example, make these changes:

   a. Specify which region server and port to connect to. For example, to run on the standalone version of Splice Machine, you could use:

   ```
   RSHostName=localhost
   SpliceConnectPort=1527
   ```

   b. Specify the user name and password for the connection. For example:

   ```
   UserName=myUserId
   UserPassword=myPassword
   ```

   c. Specify the location of the csv file in HDFS. For example:
5. **Connect to your Splice Machine database.**

6. **Create the **LINEITEM** table in your database:**

   ```sql
   CREATE TABLE LINEITEM (  
   L_ORDERKEY         INTEGER 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       CHAR(1),  
   L_LINESTATUS       CHAR(1),  
   L_SHIPDATE         DATE,  
   L_COMMITDATE       DATE,  
   L_RECEIPTDATE      DATE,  
   L_SHIPINSTRUCT     CHAR(25),  
   L_SHIPMODE         CHAR(10),  
   L_COMMENT          VARCHAR(44)--,  
   PRIMARY KEY (L_ORDERKEY, L_LINENUMBER) 
   );
   ```

7. **Run the `spark-submit.sh` script from the command line to import the data from the `lineitem.csv` file into the **LINEITEM** table in your database.**
**Best Practices: Ingesting Streaming Data**

This topic presents an example of using Spark streaming to ingest real-time data from Internet-connected devices (IoT) into a Splice Machine table via `spark-submit`. This topic includes the following sections:

- **About Ingesting Streaming Data**
- **Using the Native Spark DataSource to Ingest Streaming Data**
- **Running the App**

For an overview of best practices for data ingestion, see Best Practices: Ingesting Data, in this Best Practices chapter.

**About Ingesting Streaming Data**

Internet of Things (IoT) applications need to continually ingest data, process that data, make decisions, and then act. This decision-making pattern typically starts with an ingestion phase of streaming raw data from the edge to a storage medium; then data engineers and data scientists iteratively wrangle the data to get it into a form that can be used downstream by learning, planning, and operational systems.

The application documented in this topic shows you how to ingest streams of IoT data into Splice Machine tables. This app streams weather data from a public weather data source into a Splice Machine table that you can then use for any purpose, such as a Machine Learning application that needs to consider weather forecasts to predict critical timing of shipments.

Our demonstration app sets up a Kafka producer that streams data from a public weather service and a Kafka consumer that parses the data, transforms it into Spark DataFrames, and then uses our Native Spark DataSource to insert each DataFrame into a Splice Machine database table.

**Using the Native Spark DataSource to Ingest Streaming Data**

This section presents a sample Spark application that uses Kafka to both produce and consume a stream of real-time weather data. Coding this example involves these steps and components, each of which is described in a subsection below:

1. **Create a table** for the data in your Splice Machine database.
2. **Create a Kafka topic** for the weather data.
3. **Create a Kafka producer** to stream data.
4. **Create a Spark app** that uses Kafka to consume the stream and uses the Splice Machine Native Spark DataSource to insert the data into your database table.

**NOTE:** All of the files required to build and run this program are available here: `./examples/SparkStreamingSubmit.tar.gz`
1. Create a Table for the Data in Your Splice Machine Database
Use the following statement in your Splice Machine database to create a table:

```sql
CREATE TABLE splice.weather (
    id VARCHAR(100),
    location VARCHAR(20),
    temperature FLOAT,
    humidity FLOAT,
    time TIMESTAMP
);
```

2. Create a Kafka Topic for the Weather Data
Create your Kafka topic with a command like this:

```bash
bin/kafka-topics --create --zookeeper localhost:2181 --replication-factor 1 --partitions 2 --topic weather
```

3. Create a Kafka Producer to Stream Data
This section presents our sample code to produce a stream of weather data. The fully commented version of this code is available in `./examples/SparkStreamingSubmit.tar.gz`.

Here's the main code for our Kafka producer:
package com.splicemachine.sample;

import java.math.RoundingMode;
import java.sql.Timestamp;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.*;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;

public class KafkaTopicProducer {
    /* Static list of locations */

    public static final String[] locations = {
        "Palm Beach", "Pasco" }

    Random r = new Random();
    DecimalFormat df = new DecimalFormat("#,##");
    SimpleDateFormat sd = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSSZ");
    private String server = null;
    private long totalEvents = 1;
    private String topic = null;

    /* Adds records to a Kafka queue */
    public static void main(String[] args) throws Exception {
        KafkaTopicProducer kp = new KafkaTopicProducer();
        kp.server = args[0];
        kp.topic = args[1];
        kp.totalEvents = Long.parseLong(args[2]);
        kp.generateMessages();
    }

    Here's the code that sends messages to the Kafka queue:
*/ Sends messages to the Kafka queue. */
public void generateMessages() {
    df.setRoundingMode(RoundingMode.CEILING);

    //Define the properties for the Kafka Connection
    Properties props = new Properties();
    props.put("bootstrap.servers", server); //kafka server
    props.put("acks", "all");
    props.put("retries", 0);
    props.put("batch.size", 16384);
    props.put("linger.ms", 1);
    props.put("buffer.memory", 33554432);
    props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

    //Create a KafkaProducer using the Kafka Connection properties
    KafkaProducer<String, String> producer = new KafkaProducer<>(props);
    long nEvents = 0;

    //Loop through for the number of messages you want to put on the Queue
    for (nEvents = 0; nEvents < totalEvents; nEvents++) {
        String id = "A_" + nEvents;
        String record = id +"," + getLocation() + "," + formatDouble(getTemperature())
                       + "," + formatDouble(getHumidity()) + "," + new Timestamp(System.currentTimeMillis()).toString();
        producer.send(new ProducerRecord<String, String>(topic, id, record));
    }
    //Flush and close the queue
    producer.flush();
    producer.close();
    //display the number of messages that aw
    System.out.println("messages pushed:" + nEvents);
}

And here are the *helper* functions:
4. Create App to Consume the Stream and Insert Data into Your Table

This section presents our sample app that consumes the data stream produced by our Kafka producer and inserts it into our Splice Machine database table.

The main body of this app uses Kafka to consume entries in the stream into a Spark RDD and invokes the `doWork` method to process the stream entries. The `doWork` method:

- creates a Spark session
- connects to your Splice Machine database
- maps stream entries into a Spark DataFrame
- uses the `insert` function of the Splice Machine Native Spark DataSource to insert the data, in real-time, into the table.

**NOTE:** This code is available in `/examples/SparkStreamingSubmit.tar.gz`.

Here are the package and class import statements for the program:
package com.splicemachine.sample;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.sql.*;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.Row;
import java.io.IOException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Properties;
import com.splicemachine.derby.impl.SpliceSpark;
import com.splicemachine.spark.splicemachine.*;
import org.apache.spark.sql.types.DataTypes;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import java.util.*;
import org.apache.spark.streaming.api.java.*;
import org.apache.spark.streaming.kafka010.*;
import org.apache.kafka.common.serialization.StringDeserializer;
import scala.Tuple2;

Here's the main Kafka consumer code:
public class KafkaTopicConsumer {

    public static void main(String[] args) throws Exception {

        if(args.length < 7) {
            System.err.println("Incorrect number of params ");
            return;
        }
        final String inTargetTable = args[0];
        final String inTargetSchema = args[1];
        final String inHostName = args[2];
        final String inHostPort = args[3];
        final String inUserName = args[4];
        final String inUserPassword = args[5];
        final String kafkaBroker = args[6];
        final String kafkaTopicName = args[7];

        // Initialize Kafka config settings
        Properties props = new Properties();
        SparkConf conf = new SparkConf().setAppName("stream");
        JavaStreamingContext jssc = new JavaStreamingContext(conf, Durations.seconds(1));

        Map<String, Object> kafkaParams = new HashMap<>();
        kafkaParams.put("bootstrap.servers", kafkaBroker);
        kafkaParams.put("key.deserializer", StringDeserializer.class);
        kafkaParams.put("value.deserializer", StringDeserializer.class);
        kafkaParams.put("group.id", "test");
        kafkaParams.put("auto.offset.reset", "latest");
        kafkaParams.put("enable.auto.commit", false);
        Collection<String> topics = Arrays.asList(kafkaTopicName);

        JavaInputDStream<ConsumerRecord<String, String>> stream =
            KafkaUtils.createDirectStream(
                jssc,
                LocationStrategies.PreferConsistent(),
                ConsumerStrategies.<String, String>Subscribe(topics, kafkaParams));

        JavaPairDStream<String, String> resultRDD =
            stream.mapToPair(record -> new Tuple2<>(record.key(), record.value()));

        doWork(inTargetTable, inTargetSchema, inHostName, inHostPort,
               inUserName, inUserPassword, resultRDD, jssc);
    }

    And here's the code that moves consumed data into your Splice Machine database:
private static void doWork(String inTargetTable, String inTargetSchema, 
    String inHostName, String inHostPort, 
    String inUserName, String inUserPassword, 
    JavaPairDStream<String, String> resultRDD, 
    JavaStreamingContext jssc) throws IOException, 
    InterruptedException {

    SparkConf conf = new SparkConf();
    SparkSession spark = SparkSession.builder().appName("Reader").config(conf).getOr 
    Create();

    // Create Splice's Spark Session
    SpliceSpark.setContext(spark.sparkContext());

    String dbUrl = "jdbc:splice://" + inHostName + ":" + inHostPort 
    + "/splicedb;user=" + inUserName 
    + ";" + "password=" + inUserPassword;

    // Create a SplicemachineContext based on the provided DB connection
    SplicemachineContext splicemachineContext = new SplicemachineContext(dbUrl);

    // Set target tablename and schemaname
    String SPLICE_TABLE_ITEM = inTargetSchema + "." + inTargetTable;

    StructType schema = splicemachineContext.getSchema(SPLICE_TABLE_ITEM);
    resultRDD.foreachRDD((RDD, time) -> {
        JavaRDD<String> rdd = RDD.values();
        JavaRDD<Row> rowJavaRDD = rdd.map(new Function<String, String[]>() {
            @Override
            public String[] call(String line) throws Exception {
                return line.split(",");
            }
        }).map(new Function<String[], Row>() {
            @Override
            public Row call(String[] r) throws Exception {
                return RowFactory.create(r[0], r[1], Double.parseDouble(r[2]), 
                Double.parseDouble(r[3]),Timestamp.valueOf(r[4]));
            }
        });

        Dataset<Row> ds = spark.createDataFrame(rowJavaRDD, schema);
        splicemachineContext.insert(ds, SPLICE_TABLE_ITEM);
    });

    jssc.start();              // Start the computation
    jssc.awaitTermination();   // Wait for the computation to terminate
}
Running the App

To put it all together, you need to start streaming data, consume that data and store it in your database table, and then use the data from the table, as shown in these sections:

1. **Run the Kafka Producer to Stream Data** to start streaming weather data.
2. **Use Spark Submit to Run the App**
3. **Use the Table**

### 1. Run the Kafka Producer to Stream Data

There are actually two shell scripts involved in streaming data into our app; both are included in the .examples/SparkStreamingSubmit.tar.gz tarball:

- The `runKafkaProducer.sh` script produces a number of events for a specific Kafka stream.
- The `streamToKafka.sh` script invokes the `runKafkaProducer.sh` script a number of times, passing parameters that specify which Kafka stream to use and how many events to produce.

You run `streamToKafka` to actually start streaming data that your app will produce.

#### The `runKafkaProducer` Script

Here’s the version of the `runKafkaProducer.sh` script that is packaged into the .examples/SparkStreamingSubmit.tar.gz tarball:

```bash
#!/usr/bin/env bash
HOST="srv070"
KAFKA_LIB_DIR="/opt/cloudera/parcels/KAFKA/lib/kafka/libs/*"
java -cp target/splice-adapter-kafka-streaming-1.0-SNAPSHOT.jar:${KAFKA_LIB_DIR}\
    com.splicemachine.sample.KafkaTopicProducer\
    ${HOST}:9092 $@
```

You need to modify this script for your environment, updating at least some of the highlighted values, as appropriate.

#### The `streamToKafka` Script

The version of the `streamToKafka.sh` script in the tarball invokes the `runKafkaProducer.sh` script 2000 times, with each invocation producing 500 events:

```bash
#!/bin/bash
for ((cnt = 0; cnt < 2000; cnt++))
do
echo $cnt
sh ./runKafkaProducer.sh weather 500
sleep 2
done
```
2. Use Spark Submit to Run the App

After you've started streaming data, you can use the supplied `spark-submit.sh` script to run your app. Here's a version of this script:

```bash
#!/bin/bash

TargetTable=WEATHER
TargetSchema=SPLICE
RSHostName=srv075
SpliceConnectPort=1527
UserName=yourDBUserId
UserPassword=yourDBPassword
KafkaBroker=stl-colo-srv070:9092
KafkaTopic=weather

export SPARK_KAFKA_VERSION=0.10

spark2-submit \
--conf "spark.dynamicAllocation.enabled=false" \
--conf "spark.streaming.stopGracefullyOnShutdown=true" \
--conf "spark.streaming.concurrentJobs=1" \
--conf "spark.task.maxFailures=2" \
--conf "spark.driver.memory=4g" \
--conf "spark.driver.cores=1" \
--conf "spark.driver.extraJavaOptions=-verbose:class" \
--conf "spark.executor.extraJavaOptions=-verbose:class" \
--conf "spark.executor.extraClassPath=/etc/hadoop/conf/:/etc/hbase/conf/:/opt/cloudera/parcels/SPLICEMACHINE/lib/*:/opt/cloudera/parcels/SPARK2/lib/spark2/jars/*:/opt/cloudera/parcels/CDH/lib/hbase/lib/*" \
--conf "spark.driver.extraClassPath=/etc/hadoop/conf/:/etc/hbase/conf/:/opt/cloudera/parcels/SPLICEMACHINE/lib/*:/opt/cloudera/parcels/SPARK2/lib/spark2/jars/*:/opt/cloudera/parcels/CDH/lib/hbase/lib/*" \
--name "Spark Adapter Test" \
--class com.splicemachine.sample.KafkaTopicConsumer \
--master yarn --deploy-mode cluster --num-executors 12 --executor-memory 8G --executor-cores 4 target/splice-adapter-kafka-streaming-1.0-SNAPSHOT-jar-with-dependencies.jar \
$TargetTable $TargetSchema $RSHostName $SpliceConnectPort $UserName $UserPassword $KafkaBroker $KafkaTopic
```

Before submitting your Spark program with this script, you need to modify some of the values (at least the highlighted) at the top of the script; for our sample program, these are the values, which are summarized in the table below:

<table>
<thead>
<tr>
<th>TargetTable</th>
<th>WEATHER</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetSchema</td>
<td>SPLICE</td>
</tr>
<tr>
<td>RSHostName</td>
<td>srv075</td>
</tr>
<tr>
<td>SpliceConnectPort</td>
<td>1527</td>
</tr>
<tr>
<td>UserName</td>
<td>yourDBUserId</td>
</tr>
<tr>
<td>UserPassword</td>
<td>yourDBPassword</td>
</tr>
<tr>
<td>KafkaBroker</td>
<td>stl-colo-srv070:9092</td>
</tr>
<tr>
<td>KafkaTopic</td>
<td>weather</td>
</tr>
</tbody>
</table>
Table 1: spark-submit script variables

<table>
<thead>
<tr>
<th>Script Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetTable</td>
<td>The name of the table in your Splice Machine database into which you are importing data.</td>
</tr>
<tr>
<td>TargetSchema</td>
<td>The name of the schema in your database to which the table belongs.</td>
</tr>
<tr>
<td>RSHostName</td>
<td>The region server for connecting to your database.</td>
</tr>
<tr>
<td>SpliceConnectPort</td>
<td>The port number for connecting to your database.</td>
</tr>
<tr>
<td>UserName</td>
<td>The user name for connecting to your database.</td>
</tr>
<tr>
<td>UserPassword</td>
<td>The user password for connecting to your database.</td>
</tr>
<tr>
<td>KafkaBroker</td>
<td>The Kafka broker server.</td>
</tr>
<tr>
<td>CsvFilePath</td>
<td>The HDFS path to the CSV file you’re importing.</td>
</tr>
</tbody>
</table>

Use the Table

Once your app is running, you can query your table and use the information as you like; for example, to train a machine learning model that predicts how weather will impact delivery dates.

Here's a simple query you can use to verify that the table has been populated:

```
splice> select * from splice.weather;
```
Best Practices: Ingesting Data From an External Table

This topic describes how to ingest data stored in an external table into a Splice Machine database. For general information about using external tables, see Using External Tables.

For an overview of best practices for data ingestion, see Best Practices: Ingesting Data, in this Best Practices chapter.

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.

You can create external tables for files stored in these formats:

- ORC
- PARQUET
- Avro
- TEXTFILE

**NOTE:** 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 or Avro file.

Importing Data From an External Table

Once you’ve got an external table defined, you can import data from it into your Splice Machine database table by using the INSERT INTO statement. For example:

```
INSERT INTO myTable SELECT * FROM myExternalTable;
```

Example: Loading Data From an ORC File

This example loads data from an ORC file that is in /tmp into a table in your database, in these steps:

1. Create an external table named LINEITEM_ext in your database that points to an existing ORC file:
CREATE EXTERNAL TABLE LINEITEM_ext
(
  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)
)
STORED AS ORC
LOCATION '/tmp/lineitem/orc';

You can also link to external tables stored in the Cloud. Splice Machine currently supports access to external tables that are stored in AWS S3 buckets.

2. **Create an internal table named LINEITEM in your database to load data into:**

```sql
CREATE TABLE 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)
);
```

3. **Insert all of the data from the external table into the internal table:**

```sql
INSERT INTO LINEITEM SELECT * FROM LINEITEM_EXT;
```
Data Ingestion Parameter Values

This topic starts with a summary of the parameters used by the Splice Machine data ingestion procedures, and then provides additional details for each parameter.

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:

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table Info</td>
<td>schemaName</td>
<td>The name of the schema of the table into which to import.</td>
<td>SPLICE</td>
</tr>
<tr>
<td></td>
<td>tableName</td>
<td>The name of the table into which to import.</td>
<td>playerTeams</td>
</tr>
<tr>
<td>Data Location</td>
<td>insertColumnList</td>
<td>The names, in single quotes, of the columns to import.</td>
<td>'ID, TEAM'</td>
</tr>
<tr>
<td></td>
<td>fileOrDirectoryName</td>
<td>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.</td>
<td>/data/mydata/mytable.csv</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The SYCS_UTIL.MERGE_DATA_FROM_FILE procedure only works with single files; <strong>you cannot specify a directory name</strong> when calling SYCS_UTIL.MERGE_DATA_FROM_FILE.</td>
<td></td>
</tr>
</tbody>
</table>

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. See the [Configuring an S3 Bucket for Splice Machine Access](https://splice.com/docs/configuring-an-s3-bucket) topic for information about accessing data on S3.
<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Formats</strong></td>
<td><strong>oneLineRecords</strong></td>
<td>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.</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td><strong>charset</strong></td>
<td>The character encoding of the import file. The default value is UTF-8.</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td><strong>columnDelimiter</strong></td>
<td>The character used to separate columns. Specify null if using the comma (,) character as your delimiter.</td>
<td>'</td>
</tr>
<tr>
<td></td>
<td><strong>characterDelimiter</strong></td>
<td>The character used to delimit strings in the imported data.</td>
<td>'</td>
</tr>
<tr>
<td></td>
<td><strong>timestampFormat</strong></td>
<td>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: &quot;yyyy-MM-dd HH:mm:ss&quot;.</td>
<td>'yyyy-MM-dd HH:mm:ss.SSS'</td>
</tr>
<tr>
<td></td>
<td><strong>dateFormat</strong></td>
<td>The format of datestamps 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: &quot;yyyy-MM-dd&quot;.</td>
<td>yyyy-MM-dd</td>
</tr>
<tr>
<td></td>
<td><strong>timeFormat</strong></td>
<td>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: &quot;HH:mm:ss&quot;.</td>
<td>HH:mm:ss</td>
</tr>
<tr>
<td><strong>Problem Logging</strong></td>
<td><strong>badRecordsAllowed</strong></td>
<td>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.</td>
<td>25</td>
</tr>
</tbody>
</table>
### 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:** SPLICE

**tableName**

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

**Example:** playerTeams

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>badRecordDirectory</strong></td>
<td>The directory in which bad record information is logged. Splice Machine logs information to the <code>&lt;import_file_name&gt;.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>. On a cluster, this directory <strong>MUST be on S3, HDFS (or MapR-FS)</strong>. If you're using our Database Service product, files can only be imported from S3.</td>
<td>'importErrsDir'</td>
<td></td>
</tr>
<tr>
<td><strong>bulkImportDirectory</strong></td>
<td>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.</td>
<td>hdfs:///tmp/test_hfile_import/</td>
<td></td>
</tr>
<tr>
<td><strong>skipSampling</strong></td>
<td>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>SYCS_UTIL.BULK_IMPORT_HFILE</code> automatically determine splits for you. This parameter is only used with the <code>SYCS_UTIL.BULK_IMPORT_HFILE</code> system procedure.</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>
**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"`

**Quoting Column Names**

In the current release of Splice Machine, the individual column names in the `insertColumnList` do not need to be quoted, even if they contain special characters. However, if you **do** quote any column name, you must quote all of the column names.

For example, the following are both valid column list specifications:

```
'ID, TEAM, #MYCOLNAME'
"ID", "TEAM", "MYCOLNAME"
```

BUT, the following column list will generate an error:

```
'ID, TEAM, "MYCOLNAME"
```

**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.IMPORT_DATA` procedure, 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*](#) below 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:

<table>
<thead>
<tr>
<th>Special character</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>'\t'</code></td>
<td>Tab</td>
</tr>
<tr>
<td><code>'\f'</code></td>
<td>Formfeed</td>
</tr>
<tr>
<td><code>'\b'</code></td>
<td>Backspace</td>
</tr>
<tr>
<td><code>'\'</code></td>
<td>Backslash</td>
</tr>
<tr>
<td><code>'^a'</code> (or <code>'^A'</code>)</td>
<td>Control-a</td>
</tr>
</tbody>
</table>

**Notes:**
- To use the single quote ('') character as your column 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: `''`

**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
'^a'
(or '^A') | Control-a
'\'' | Single Quote (')

**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.

**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:**
```
'''
```

**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:
<table>
<thead>
<tr>
<th>Pattern Letter</th>
<th>Description</th>
<th>Format(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>year</td>
<td>yy or yyyy</td>
</tr>
<tr>
<td>M</td>
<td>month</td>
<td>MM</td>
</tr>
<tr>
<td>d</td>
<td>day in month</td>
<td>dd</td>
</tr>
<tr>
<td>a</td>
<td>Am/pm marker (AM or PM)</td>
<td>a or aa</td>
</tr>
<tr>
<td>h</td>
<td>hour (0-12)</td>
<td>hh</td>
</tr>
<tr>
<td>H</td>
<td>hour (0-23)</td>
<td>HH</td>
</tr>
<tr>
<td>m</td>
<td>minute in hour</td>
<td>mm</td>
</tr>
<tr>
<td>s</td>
<td>seconds</td>
<td>ss</td>
</tr>
<tr>
<td>S</td>
<td>tenths of seconds</td>
<td>s, ss, sss, ss, sssss or sssssss*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*Specify sssssss to allow a variable number (any number) of digits after the decimal point.</td>
</tr>
<tr>
<td>z</td>
<td>time zone text</td>
<td>e.g. Pacific Standard time</td>
</tr>
<tr>
<td>Z</td>
<td>time zone, time offset</td>
<td>e.g. -0800</td>
</tr>
</tbody>
</table>

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:
<table>
<thead>
<tr>
<th>Timestamp value</th>
<th>Format Pattern</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-03-23</td>
<td>yyyy-MM-dd HH:mm:ss</td>
<td>This is the default pattern.</td>
</tr>
<tr>
<td>2013-03-23</td>
<td>yyyy-MM-dd HH:mm:ss.SSZ</td>
<td>This pattern allows up to 2 decimal digits of seconds, and requires a time zone specification.</td>
</tr>
<tr>
<td>2013-03-23</td>
<td>yyyy-MM-dd HH:mm:ssZ</td>
<td>This patterns requires a time zone specification, but does not allow for decimal digits of seconds.</td>
</tr>
<tr>
<td>2013-03-23</td>
<td>yyyy-MM-dd HH:mm:ss.123</td>
<td>This pattern allows up to 3 decimal digits of seconds, but does not allow a time zone specification.</td>
</tr>
<tr>
<td>2013-03-23</td>
<td>yyyy-MM-dd HH:mm:ss.12</td>
<td>Note that if your data specifies more than 3 decimal digits of seconds, an error occurs.</td>
</tr>
<tr>
<td>2013-03-23</td>
<td>yyyy-MM-dd HH:mm:ss.1298</td>
<td>This pattern allows up to 4 decimal digits of seconds, but does not allow a time zone specification.</td>
</tr>
</tbody>
</table>

See [Time and Date Formats in Input Records](#) below 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](#) below 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](#) below 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 SYCS_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/'
If you’re using this procedure with our On-Premise database product, on a cluster with Cloudera Key Management Service (KMS) enabled, there are a few extra configuration steps required. Please see this troubleshooting note for details.

**skipSampling**

**NOTE:** This parameter is only used with the SYCS_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 `SYCS_UTIL.SYCS_SPLIT_TABLE_OR_INDEX` system procedure to compute splits for your table before calling `SYCS_UTIL.BULK_IMPORT_HFILE`. This allows you more control over the splits, but adds a layer of complexity. You can learn about computing splits for your input data in the Bulk Importing Flat Files topic of this chapter.

- **If `skipSampling` is false**, then `SYCS_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

**Example:** false

### Additional Information About Ingestion Parameters

This section contains the following details about how certain parameters are used by the Splice Machine ingestion procedures:

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

**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/mynname/mydatadir/mytable.csv
```

**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/mynname/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:

```java
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:

```java
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:

```xml
<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 Formats**

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`

If you are importing a lot of data, our `SYSCS_UTIL.BULK_IMPORT_HFILE` bulk import procedure greatly improves data loading performance by splitting the data into HFiles, doing the import, and then deleting the HFiles. You can have `SYCS_UTIL.BULK_IMPORT_HFILE` use sampling to determine the keys to use for splitting your data by, or you can use the `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX` procedure to compute the splits, and then call the bulk import procedure. For more information, see the Best Practices: Bulk Importing Flat Files topic in this chapter.

---

**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:

<table>
<thead>
<tr>
<th>Special character</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>\t</td>
<td>Tab</td>
</tr>
<tr>
<td>\f</td>
<td>Formfeed</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
<tr>
<td>^a (or ^A)</td>
<td>Control-a</td>
</tr>
</tbody>
</table>

**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

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 About Date, Time, and Timestamp Values
A few additional notes:

- 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:
**Inserting and Updating Column Values When Importing Data**

This section summarizes what happens when you are importing 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 `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 `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 the `IMPORT_DATA` and `MERGE_DATA_FROM_FILE` procedures:

<table>
<thead>
<tr>
<th>Column included in <code>importColumnList</code>?</th>
<th>Table column conditions</th>
<th>Action Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>N/A</td>
<td>Import value inserted into table column if valid; if not valid, a bad record error is logged.</td>
</tr>
<tr>
<td>NO</td>
<td>Has Default Value</td>
<td>Default value is inserted into table column.</td>
</tr>
<tr>
<td></td>
<td>Is Generated Value</td>
<td>Generated value is inserted into table column.</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>NULL is inserted into table column.</td>
</tr>
</tbody>
</table>

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:

```sql
CREATE TABLE myTable (  
    colA INT,  
    colB CHAR(12) DEFAULT 'myDefaultVal',  
    colC INT);  
```
### 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`. 
When using the MERGE_DATA_FROM_FILE procedure to update an existing record in a table: columns not listed in your importColumnList are not updated.

**Importing CLOBs and BLOBs**

When importing CLOBs, be sure to review these tips to avoid common problems:

- Be sure that the data you’re importing does not include a special character that you’ve designated as a column or character delimiter. Otherwise, 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.

- If your data contains line feed characters like CTRL-M, you need to set the oneLineRecords parameter to false to allow Splice Machine to properly handle the data; 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.

At this time, the Splice Machine import procedures do not work with columns of type BLOB. You can, however, 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:

```sql
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.
Best Practices: Troubleshooting On-Premise Bulk Import

This section contains the following tips for troubleshooting ingestion of data into our On-Premise Database product:

- **Using Bulk Import on a KMS-Enabled Cluster**
- **Bulk Import: Out of Heap Memory**
- **Bulk Import: Out of Direct Memory (Cloudera)**
- **Bulk Import: Network Timeout**

There are currently no troubleshooting issues to address if you're using the Splice Machine Database-as-Service product.

**Using Bulk Import on a KMS-Enabled Cluster**

If you are a Splice Machine On-Premise Database customer and want to use bulk import on a cluster with Cloudera Key Management Service (KMS) enabled, you must complete these extra configuration steps:

1. Make sure that the `bulkImportDirectory` is in the same encryption zone as is HBase.
2. Add these properties to `hbase-site.xml` to load secure Apache BulkLoad and to put its staging directory in the same encryption zone as HBase:

   ```xml
   <property>
     <name>hbase.bulkload.staging.dir</name>
     <value><YourStagingDirectory></value>
   </property>
   <property>
     <name>hbase.coprocessor.region.classes</name>
     <value>org.apache.hadoop.hbase.security.access.SecureBulkLoadEndpoint</value>
   </property>
   ```

   Replace `<YourStagingDirectory>` with the path to your staging directory, and make sure that directory is in the same encryption zone as HBase; for example:

   ```xml
   <value>/hbase/load/staging</value>
   ```

For more information about KMS, see [https://www.cloudera.com/documentation/enterprise/latest/topics/cdh_sg_kms.html](https://www.cloudera.com/documentation/enterprise/latest/topics/cdh_sg_kms.html).
Bulk Import: Out of Heap Memory
If you run out of heap memory while bulk importing an extremely large amount of data with our On-Premise product, you can resolve the issue by setting the hbase client's hfile.block.cache.size property value to a very small number. We recommend this setting:

```
hfile.block.cache.size=0.01
```

**NOTE:** This setting should be applied only to the HBase client.

Bulk Import: Network Timeout
If you encounter a network timeout during bulk ingestion with our On-Premise product, you can resolve it by adjusting the value of the shuffle.io.connectionTimeout property as follows:

```
-Dsplice.spark.shuffle.io.connectionTimeout=480s
```

Bulk Import: Out of Direct Memory (Cloudera)
When using the On-Premise version of Splice Machine with Spark with Cloudera, bulk import of very large datasets can fail due to direct memory usage. Use the following settings to resolve this issue:

1. Update the Shuffle-to-Mem Setting
   Modify the following setting in the Cloudera Manager’s Java Configuration Options for HBase Master:

   ```
   -Dsplice.spark.reducer.maxReqSizeShuffleToMem=134217728
   ```

2. 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_FIRST=true
   ```

3. Due to how Yarn manages memory, you need to modify your YARN configuration when bulk-importing large datasets. Make this change in your Yarn configuration, ResourceManager Advanced Configuration Snippet (Safety Valve) for yarn-site.xml:

   ```
   yarn.nodemanager.vmem-check-enabled=false
   ```
You may also need to **temporarily** make this additional configuration update as a workaround for memory allocation issues. Note that this update **is not recommended for production usage**, as it affects all YARN jobs and could cause your cluster to become unstable:

```
yarn.nodemanager.pmem-check-enabled=false
```
Importing the TPCH Data into Your Database

This topic shows you how to import the TPCH Data into your Splice Machine database, and then includes the SQL for the TPCH queries.

You can use the following steps to import TPCH data into your 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_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)  
);

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. See the [Configuring an S3 Bucket for Splice Machine Access](#) topic for information about accessing data on S3.
You can copy/paste the following `SYSCS_UTIL.IMPORT_DATA` statements to quickly pull that data into your database:

```sql
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);
```

You need to supply your AWS credentials in each URL or in your `core-site.xml` configuration file to read data from S3, as described here.

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_linestatus,
    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_linestatus
order by
    l_returnflag,
    l_linestatus
-- 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.

The TPCH Queries
Here are a number of additional queries you might want to run against the TPCH data:
TPCH Query 01

```
-- QUERY 01
select
    l_returnflag,
    l_linestatus,
    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
    lineitem
where
    l_shipdate <= date({fn TIMESTAMPADD(SQL_TSI_DAY, -90, cast('1998-12-01 00:00:00' as timestamp)})
order by
    l_returnflag,
    l_linestatus
-- END OF QUERY
```
TPCH Query 02

-- QUERY 02
select
    s_acctbal,
    s_name,
    n_name,
    p_partkey,
    p_mfgr,
    s_address,
    s_phone,
    s_comment
from
    part,
    supplier,
    partsupp,
    nation,
    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
        partsupp,
        supplier,
        nation,
        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}
-- END OF QUERY
;
-- QUERY 03

select
    l_orderkey,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    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_shippriority

    order by
        revenue desc,
        o_orderdate

{limit 10}

-- END OF QUERY
;

-- QUERY 04
select
    o_orderpriority,
    count(*) as order_count
from
    orders
where
    o_orderdate >= date('1993-07-01')
and o_orderdate < add_months('1993-07-01',3)
and exists (
    select
        *
    from
        lineitem
    where
        l_orderkey = o_orderkey
    and l_commitdate < l_receiptdate
)
group by
    o_orderpriority
order by
    o_orderpriority
-- END OF QUERY
;
TPCH Query 05

```sql
-- QUERY 05
select
    n_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue
from
    customer,
    orders,
    lineitem,
    supplier,
    nation,
    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
```

TPCH Query 06

```sql
-- QUERY 06
select
    sum(l_extendedprice * l_discount) as revenue
from
    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
```
TPCH Query 07

-- 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
        supplier,
        lineitem,
        orders,
        customer,
        nation n1,
        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')
    ) as shipping
group by
    supp_nation,
    cust_nation,
    l_year
order by
    supp_nation,
    cust_nation,
    l_year
-- END OF QUERY ;
TPCH Query 08

```
-- 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 part,
        supplier,
        lineitem,
        orders,
        customer,
        nation n1,
        nation n2,
        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('1996-12-31')
        and p_type = 'ECONOMY ANODIZED STEEL'
    ) as all_nations
group by
    o_year
order by
    o_year
-- END OF QUERY
;```
TPCH Query 09

-- 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_quantity
    as amount
    from
        part,
        supplier,
        lineitem,
        partsupp,
        orders,
        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
;
-- 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
  customer,
  orders,
  lineitem,
  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
;
-- QUERY 11
select
    ps_partkey,
    sum(ps_supplycost * ps_availqty) as value
from
    partsupp,
    supplier,
    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) * 0.0000010000
        from
            partsupp,
            supplier,
            nation
        where
            ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = 'GERMANY'
    )
order by
    value desc
-- END OF QUERY
-- 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
    orders,
    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))})
    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
;
TPCH Query 13

```sql
-- QUERY 13
select
    c_count,
    count(*) as custdist
from
    (select
        c_custkey,
        count(o_orderkey)
    from
        customer left outer join 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
;
```

TPCH Query 14

```sql
-- 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
    lineitem,
    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
;
```
TPCH Query 15

-- QUERY 15
create view revenue0 (supplier_no, total_revenue) as
   select
       l_suppkey,
       sum(l_extendedprice * (1 - l_discount))
   from
       lineitem
   where
       l_shipdate >= date('1996-01-01')
       and l_shipdate < add_months('1996-01-01',3)
   group by
       l_suppkey;

select
    s_suppkey,
    s_name,
    s_address,
    s_phone,
    total_revenue
from
    supplier,
    revenue0
where
    s_suppkey = supplier_no
    and total_revenue = (select max(total_revenue) from revenue0)
order by
    s_suppkey;

drop view revenue0
-- END OF QUERY
;
TPCH Query 16

-- QUERY 16
select
    p_brand,
    p_type,
    p_size,
    count(distinct ps_suppkey) as supplier_cnt
from
    partsupp,
    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
        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
;
TPCH Query 17

```sql
-- QUERY 17
select
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem, 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
        lineitem
    where
        l_partkey = p_partkey
)
-- END OF QUERY
;
```
TPCH Query 18

-- QUERY 18

select
c_name,
c_custkey,
o_orderkey,
o_orderdate,
o_totalprice,
sum(l_quantity)
from
customer,
orders,
lineitem
where
  o_orderkey in (
    select
      l_orderkey
    from
      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
;
select
  sum(l_extendedprice* (1 - l_discount)) as revenue
from
  lineitem, part
where
  (p_partkey = l_partkey
  and p_brand = 'Brand#12'
  and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
  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 PACK')
  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 PKG')
  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
;
-- QUERY 20
select
    s_name,
    s_address
from
    supplier,
    nation
where
    s_suppkey in ( select
                    ps_suppkey
                    from
                    partsupp
                    where
                    ps_partkey in ( select
                                    p_partkey
                                    from
                                    part
                                    where
                                    p_name like 'forest%'
                                )
                )
    and ps_availqty > ( select
                        0.5 * sum(l_quantity)
                        from
                        lineitem
                        where
                        l_partkey = ps_partkey
                        and l_suppkey = ps_suppkey
                        and 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 s_nationkey = n_nationkey
    and n_name = 'CANADA'
order by
    s_name
-- END OF QUERY
;
-- QUERY 21

select
    s_name,
    count(*) as numwait
from
    supplier,
    lineitem l1,
    orders,
    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
        lineitem l2
    where
        l2.l_orderkey = l1.l_orderkey
        and l2.l_suppkey <> l1.l_suppkey
    )
and not exists (  
    select
        *
    from
        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
;
-- QUERY 22
select
  cntrycode,
  count(*) as numcust,
  sum(c_acctbal) as totacctbal
from
  (select
    SUBSTR(c_phone, 1, 2) as cntrycode,
    c_acctbal
  from
    customer
  where
    SUBSTR(c_phone, 1, 2) in ('13', '31', '23', '29', '30', '18', '17')
  and c_acctbal > (select
                          avg(c_acctbal)
                      from
                          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
                          orders
                      where
                          o_custkey = c_custkey)
) as custsale
group by
  cntrycode
order by
  cntrycode
-- END OF QUERY
;
Best Practices for Updating Your Splice Machine Software

This section contains best practice and troubleshooting information related to updating your Splice Machine On-Premise Database product software, in these topics:

- Updating Stored Query Plans after a Splice Machine Update

This is an On-Premise-Only topic! Learn more

Updating Stored Query Plans after a Splice Machine Update

When you install a new version of your Splice Machine software, you may need to make these calls:

```sql
CALL SYSCS_UTIL.SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE();
CALL SYSCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS();
CALL SYSCS_UTIL.SYSCS_UPDATE_METADATA_STORED_STATEMENTS();
```

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.
Understanding and Troubleshooting Splice Machine Backups

This section contains best practice and troubleshooting information related to backing up your data with our On-Premise Database product, in these topics:

- How Backup Jobs Run
- Configuring Backups When Copying with Spark Executors
- Configuring Backups When Copying with distcp

This is an On-Premise-Only topic! Learn more

How Backup Jobs Run
Splice Machine backups run as Spark jobs, submitting tasks to copy HFiles.

In the past, Splice Machine backups used the Apache Hadoop distcp tool to copy the HFile; distcp uses MapReduce to copy, which can require significant resources. These requirements can limit file copying parallelism and reduce backup throughput. Splice Machine backups now can run (and do so by default) using a Spark executor to copy the HFiles, which significantly increases backup performance. You can choose to revert to using distcp, as described in the Resource Configuration When Using distcp section below.

Configuring Backups When Copying with Spark Executors
The default way for backups to run is using Spark executors to perform the file copies, which results in a significant performance boost compared to the older method of using distcp. You can configure these backups using the following options:

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>splice.backup.max.bandwidth.mb</td>
<td>100</td>
<td>Sets the maximum throughput, in megabytes, for one thread of file copying; if copying is faster than this value, I/O is throttled to avoid consuming too much network and disk bandwidth. The default value is 100MB per second.</td>
</tr>
<tr>
<td>splice.backup.io.buffer.size</td>
<td>64</td>
<td>This is the size, in kilobytes, of the buffer used to read from and write to HDFS. Tuning this value has an impact of backup performance. The default value is 64KB.</td>
</tr>
</tbody>
</table>
Resource Configuration When Using distcp

You can choose to have Splice Machine use distcp for copying HFiles when backing up your database. To do so, set the following configuration option:

splice.backup.use.distcp = true

The default value of splice.backup.use.distcp is false.

When using distcp for copying, 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.

\[(\text{splice.spark.dynamicAllocation.minExecutors} + 1) < (\text{yarn.nodemanager.resource.cpu-vcores} \times \text{number_of_nodes})\]

\[(\text{splice.spark.dynamicAllocation.minExecutors} \times (\text{splice.spark.yarn.executor.memoryOverhead} + \text{splice.spark.yarn.executor.memory}) + \text{splice.spark.yarn.am.memory}) < (\text{yarn.nodemanager.resource.memory-mb} \times \text{number_of_nodes})\]

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:

\[\text{yarn.app.mapreduce.am.resource.cpu-vcores} \times \text{splice.backup.parallelsim}\]

There must be at least this many additional vcores available to execute Map Reduce tasks:

\[\max\{\text{mapreduce.map.cpu.vcores}, \text{mapreduce.reduce.cpu.vcores}\}\]

Thus, the total number of vcores that must be available for Map Reduce jobs is:

\[\text{yarn.app.mapreduce.am.resource.cpu-vcores} \times \text{splice.backup.parallelsim} + \max\{\text{mapreduce.map.cpu.vcores}, \text{mapreduce.reduce.cpu.vcores}\}\]
**Verify that adequate memory is available**

The Map Reduce application master requires this much memory:

\[
\text{yarn.scheduler.minimum-allocation-mb } \times \text{splice.backup.parallelim}
\]

There must be at least this much memory available to execute Map Reduce tasks:

\[
\text{yarn.scheduler.minimum-allocation-mb}
\]

Thus, the total number of memory that must be available for Map Reduce jobs is:

\[
\text{yarn.scheduler.minimum-allocation-mb } \times (\text{splice.backup.parallelim}+1)
\]
Splice Machine Best Practices for Database Security

This section contains best practice and troubleshooting information related to configuring security in our *On-Premise Database* product, in these topics:

- Assigning Full Administrative Privileges with LDAP
- Kerberos Configuration Option
- Assigning Full Administrative Privileges with Native and Kerberos Authentication

Assigning Full Administrative Privileges with LDAP

The default administrative user ID in Splice Machine is *splice*. If you're using LDAP and want to configure other users to have the same privileges as the *splice* user, follow these steps:

1. Add an LDAP admin_group mapping for the *splice* user to both:
   - the HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml
   - the HBase Client Advanced Configuration Snippet (Safety Valve) for hbase-site.xml

   This maps members of the admin_group to the same privileges as the *splice* user:

   ```
   <property>
     <name>splice.authentication.ldap.mapGroupAttr</name>
     <value>admin_group=splice</value>
   </property>
   ```

2. Assign the admin_group to a user by specifying cn=admin_group in the user definition. For example, we'll add a *myUser* with these attributes:

   ```
   <div class="preWrapperWide"><pre class="Example">
   dn: cn=admin_group,ou=Users,dc=splicemachine,dc=colo
   sn: MyUser
   objectClass: inetOrgPerson
   userPassword: myPassword
   uid: myUser
   </pre></div>
   ```

   You need to make sure that the name you specify for cn is exactly the same as the name (the value to the left of the equality symbol) in the splice.authentication.ldap.mapGroupAttr property value. Matching is case sensitive!

Now *myUser* belongs to the admin_group group, and thus gains all privileges associated with that group.
Verifying the Configuration

We can now run a few tests to verify the the super user myUser has administrative privileges. Suppose:

- userA and userB are regular LDAP users
- userA owns the schema userA in your Splice Machine database
- userB owns the schema userB in your Splice Machine database
- the userA schema has a table named t1, the contents of which are shown below
- the userB schema has a table named t2, the contents of which are shown here:

```
splice> select * from userA.t1
T1
------------
1
2
3

3 rows selected
splice> select * from userB.t2
T2
------------
1
2
3

3 rows selected
```

Now we'll run two tests using the splice> command line interpreter:

**Test 1: Verify that myUser can access schemas and tables belonging to both**

1. Connect to Splice Machine as myUser:

   ```
   connect 'jdbc:splice://localhost:1527/splicedb;user=myUser;password=myUserPassWor
d' as myuser_con;
   ```

2. Verify that you can select from both schemas:
splice select * from userA.t1
T1
-------------
1
2
3
3 rows selected
splice> select * from userB.t2
T2
-------------
1
2
3
3 rows selected

3. Make sure that while connected to myuser_con, you can also perform table operations such as insert, delete, update, and drop table. Also make sure you can create and drop schemas.

**Test 2: Verify that myUser can grant privileges to other users.**

We'll test this by granting privileges on schema userB to userA and confirming that userA can access the schema.

splice> connect 'jdbc:splice://localhost:1527/splicedb;user=userA;password=userAPassword' as userA_con;
splice> select * from userB.t2;
ERROR 42502: User 'USERA' does not have SELECT permission on column 'A2' of table 'USERB.'T2'.
splice> set connection myuser_con;
splice> grant all privileges on schema userB to userA;
0 rows inserted/updated/deleted
splice> set connection userA_con;
splice> select * from userB.t2;
A2
-------------
1
2
3
3 rows selected

**Assigning to Multiple Groups**

You can assign the privileges to multiple groups by specifying a comma separated list in the ldap.mapGroupAttr property. For example, changing its definition to this:

```
<property>
  <name>splice.authentication.ldap.mapGroupAttr</name>
  <value>admin_group=splice,cdl_group=splice</value>
</property>
```
Means that members of both the admin_group and cd1_group groups will have the same privileges as the splice user.

**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
```

**Assigning Full Administrative Privileges with Native and Kerberos Authentication**

The default administrative user ID in Splice Machine is splice. If you're using Native Authentication and Kerberos Authentication, and you want to configure other users to have the same privileges as the splice user, you need to set the mapping between splice and the other user(s) in the splice.authentication.ldap.mapGroupAttr configuration parameter. Here’s an example:

1. Follow the steps in the Enabling Kerberos Authentication topic to add a new principal; for this example, we’ll assume the new principal is named jdoe.

2. Assign admin privileges to jdoe by setting the splice.authentication.ldap.mapGroupAttr property as follows:

   ```
   <property>
     <name>splice.authentication.ldap.mapGroupAttr</name>
     <value>jdoe=splice</value>
   </property>
   ```

3. Log into your database as jdoe:

   ```
   connect 'jdbc:splice://regionsevername:1527/splicedb;principal=jdoe@SPLICEMACHINE.C0LO;keytab=jdoe_filepath.keytab' as jdoe_con;
   ```

4. check that jdoe has group user “SPLICE“ and he can create and drop schemas
splice> values user;
1
------------------------
JDOE
1 row selected
splice> values group_user;
1
------------------------
"SPLICE"
1 row selected
Configuring Your Splice Machine Database Installation

The following table provides summary information about (some of) the configuration options used by Splice Machine.

<table>
<thead>
<tr>
<th>Property</th>
<th>Typical value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>splice.authentication</td>
<td>NATIVE</td>
<td>This is documented in our Configuring Authentication topic.</td>
</tr>
<tr>
<td>splice.authentication.native.algorithm</td>
<td>SHA-512</td>
<td>This is documented in our Configuring Authentication topic.</td>
</tr>
<tr>
<td>splice.client.write.maxDependentWrites</td>
<td>60000</td>
<td>A form of write throttling that controls the maximum number of concurrent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dependent writes from a <em>single process</em>. Dependent writes are writes to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a table with indexes and generate more independent writes (writes to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>indexes themselves). They are segregated so we can guarantee progress by</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reserving some IPC threads for independent writes.</td>
</tr>
</tbody>
</table>

If your application is experiencing unexpected timeouts, see the *Adjusting for Timeouts* section below.
<table>
<thead>
<tr>
<th>Property</th>
<th>Typical value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>splice.client.write.maxIndependentWrites</td>
<td>60000</td>
<td>A form of write throttling that controls the maximum number of concurrent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>independent writes from <strong>a single process</strong>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If your application is experiencing unexpected timeouts, see <strong>Adjusting</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>for Timeouts</strong> section below.</td>
</tr>
<tr>
<td>splice.client.writer.maxThreads</td>
<td>200</td>
<td>The total number of threads for the writer in the process, which determines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the maximum number of concurrent writes from a process.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If your application is experiencing unexpected timeouts, see <strong>Adjusting</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>for Timeouts</strong> section below.</td>
</tr>
<tr>
<td>splice.compression</td>
<td>snappy</td>
<td>The type of compression to use when compressing Splice Tables. This is set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the same HBase sets table compression, and has the same codecs available to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>it (GZIP,Snappy, or LZO depending on what is installed).</td>
</tr>
<tr>
<td>splice.debug.logStatementContext</td>
<td>false</td>
<td>Whether to log all statements. Note that this is costly in OLTP workloads.</td>
</tr>
<tr>
<td>splice.marshal.kryoPoolSize</td>
<td>1100</td>
<td>The maximum number of Kryo objects to pool for reuse. This setting should</td>
</tr>
<tr>
<td></td>
<td></td>
<td>only be adjusted if there are an extremely large number of operations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed on the system.</td>
</tr>
</tbody>
</table>
### Property Table

<table>
<thead>
<tr>
<th>Property</th>
<th>Typical value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>splice.olap_server.clientWaitTime</td>
<td>900000</td>
<td>The number of milliseconds the OLAP client should wait for a result.</td>
</tr>
<tr>
<td>splice.root.path</td>
<td>/splice</td>
<td>Zookeeper root node for Splice Machine. All Zookeeper nodes used by Splice Machine will hang from this root node.</td>
</tr>
<tr>
<td>splice.splitBlockSize</td>
<td>67108864</td>
<td>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 depend on a number of factors, including the number of storefiles in use by a given HBase region.</td>
</tr>
<tr>
<td>splice.timestamp_server.clientWaitTime</td>
<td>120000</td>
<td>The number of milliseconds the timestamp client should wait for a result.</td>
</tr>
<tr>
<td>splice.txn.completedTxns.concurrency</td>
<td>128</td>
<td>Concurrency level for the completed transactions cache. Specifies the estimated number of concurrently updating threads.</td>
</tr>
<tr>
<td>splice.txn.concurrentLevelLevel</td>
<td>4096</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

**Adjusting for Application Timeouts**

If your application is experiencing unexpected timeouts, it may be due to the queue getting too large, which means that you need to reduce throughput by adjusting these property values:

- `splice.client.write.maxIndependentWrites`
- `splice.client.write.maxDependentWrites`
- `splice.client.writer.maxThreads` values

These values apply *per process*.

Here are some guidelines:

- The `splice.client.write.maxIndependentWrites` and `splice.client.write.maxDependentWrites`
maximum concurrent write values must be within the capacity of the nodes in your cluster.

You should set the value of the `splice.client.writer.maxThreads` property as follows:

\[
\text{num-nodes} \times \text{num-executors-per-node} \times (\text{max-independent-write-threads} + \text{max-dependent-write-threads})
\]

Make sure that the values you select for these properties are commensurate with your read-side throughput.
Configuring Performance of Your Splice Machine Database

This section contains best practice and troubleshooting information related to modifying configuration options to fine-tune database performance with our On-Premise Database product, in these topics:

- Resolving Periodic Spikes in HBase Read Times
- Increasing Parallelism for Spark Shuffles
- Increasing Memory Settings for Heavy Analytical Work Loads
- Force Compaction to Run Locally

This is an On-Premise-Only topic!

Resolving Periodic Spikes in HBase Read Times

If you’re using Cloudera and you closely monitor your read request queues as a way to stay on top of your cluster load, you might observe a spike in reads every 30 minutes. Cloudera Manager enables an HBase Region Health Canary that pings every server once every 30 minutes. As long as you are not experiencing any throughput problems, these spikes are harmless. If you want to get rid of the spikes, you can disable this monitoring, as follows:

1. In Cloudera Manager, navigate to HBase service -> Configuration -> Monitoring.
2. Deselect (uncheck) HBase Region Health Canary.

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.

Increasing Memory Settings for Heavy Analytical Work Loads

If you are running heavy analytical loads or running OLAP jobs on very large tables, you may want to increase these property settings in your hbase-site.xml file:
<table>
<thead>
<tr>
<th>Property</th>
<th>Default Value (MB)</th>
<th>Recommendations for Heavy Analytical Loads</th>
</tr>
</thead>
<tbody>
<tr>
<td>splice.olap_server.memory</td>
<td>1024</td>
<td>Set to the same value as HMaster heap size</td>
</tr>
<tr>
<td>splice.olap_server.memoryOverhead</td>
<td>512</td>
<td>Set to 10% of splice.olap_server.memory</td>
</tr>
<tr>
<td>splice.olap_server.virtualCores</td>
<td>1 vCore</td>
<td>4 vCores</td>
</tr>
<tr>
<td>splice.olap_server.external</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

**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
```
Best Practices for Maintaining Your Splice Machine Software

This section contains best practice and troubleshooting information related to maintaining your Splice Machine *On-Premise Database* product software, in these topics:

- Managing Your Disk Space

Managing Your Disk Space

Make sure to keep an eye on disk space. Yarn automatically refuses to start applications, including the Splice Spark and OlapMaster applications, when the *disk threshold* is crossed. That threshold is set to 90% by default; Splice Machine **strongly recommends against** raising the threshold, because disks slow down as a function of their fullness, due to fragmentation; this could lead to your database ceasing to function.
Restarting Your Splice Machine Database

This section contains best practice and troubleshooting information related to restarting Splice Machine after a forced or unexpected shutdown, in these topics:

- **Restarting Splice Machine after an HMaster Failure**
- **Slow Restart After Forced Shutdown**

This is an On-Premise-Only topic!  
[Learn more](#)

### 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

### Slow Restart After Forced Shutdown

We have seen a situation where HMaster doesn't exit when you attempt a shutdown, and a forced shutdown is used. The forced shutdown means that HBase may not be able to flush all data and delete all write-ahead logs (WALs); as a result, it can take longer than usual to restart HBase and Splice Machine.

Splice Machine now sets the HBase *Graceful Shutdown Timeout* to 10 minutes, which should be plenty of time. If the shutdown is still hanging up after 10 minutes, a forced shutdown is appropriate.
Overview of the Splice Machine Native Spark DataSource

This topic provides general information about the Splice Machine Native Spark DataSource, aka the Splice Machine Spark Adapter.

The other topics in this chapter provide additional information about the Native Spark DataSource:

- Native Spark DataSource API provides reference information for the Native Spark DataSource API methods.
- Native Spark DataSource Examples includes examples that show you how to launch a Spark app with our Spark Submit script, and how to use the Native Spark DataSource interactively, with the Spark Shell.
- Using Our Native Spark DataSource with Zeppelin presents an example of using our Native Spark DataSource in a Zeppelin notebook.

Native Spark DataSource Overview

The Splice Machine Native Spark DataSource, which is also referred to as the Spark Adapter, allows you to directly connect Spark DataFrames and Splice Machine database tables, bypassing the need to send your data to/from Spark over a JDBC or ODBC connection. You can efficiently insert, upsert, select, update, and delete data in your Splice Machine tables directly from Spark in a transactionally consistent manner. With the Spark Adapter, transfers of data between Spark and your database are completed without serialization/deserialization, which generates tremendous performance boosts over traditional over-the-wire (sequentially over a JDBC/ODBC connection) transfers.

To use the Spark Adapter in your code, you simply instantiate a SplicemachineContext object in your Spark code. You can run Spark applications that interface with your Splice Machine database interactively in the Spark shell, in Zeppelin notebooks, or by using our Spark Submit script. One common use of the Adapter is to ingest data into your Splice Machine database directly from a Spark DataFrame.

The Native DataSource allows data scientists to bypass the limitations of the SQL-based JDBC interface in favor of the more scalable and powerful Spark DataFrame API, making it possible for them to operate on data at scale and ingest real-time streaming data with outstanding performance. You can craft applications that use Spark and our Native Spark DataSource in Scala, Python, and Java. Note that you can use the Native Spark DataSource in the Splice Machine ML Manager and Zeppelin Notebook interfaces.

Why Use the Native DataSource?

The primary reason for using the Native DataSource is that it provides dramatic performance improvements for large scale data operations; this is because the DataSource works directly on native DataFrames and RDDs, thus eliminating the need to serialize data. Spark is optimized to work on DataFrames, which is a distributed collection of data (an RDD) organized into named columns, with a schema that specifies data types, that is designed to support efficiently operating on scalable, massive datasets.

The Splice Machine DataSource is native to Spark, which means that it operates directly on these DataFrames and in the same Spark executors that your programs are using to analyze or transform the data. Instead of accessing, inserting, or manipulating data one record at a time over a serialized connection, you can use the Splice Machine Native Spark DataSource to pull the contents of an entire DataFrame into your database, and to pull database query results into a DataFrame.
Splice Machine has observed 100x performance increases compared to using JDBC for operations such as inserting millions of records in a database! For example, a typical web application might use a JDBC connection to query the database, pulling information out one record at a time to populate the screen. The results of each query are serialized (turned into a string of data), then sent over a network connection to the app, and then displayed on the customer's screen.

In contrast, when you use the Splice Machine Native Spark DataSource, the contents of the database table are typically sitting in a DataFrame in memory that resides on the same Spark executor that's performing the query. The query takes place in memory, and there's no need to serialize or stream the results over a wire. Similarly, when the app sends updates to the database, the data is inserted into the database from in-memory DataFrames directly to the tables without serialization. As a result, a great deal of overhead is eliminated, and performance gains can be remarkable.

**Leveraging Developer Agility**

The Native Spark DataSource provides support for the development tools of the Data Scientists and Data Engineers alike. Data Scientists and Data Engineers typically access Spark contexts to operate on DataFrames. Spark provides a powerful set of transformations and actions to the developer to manipulate large datasets efficiently plus additional libraries for machine learning and streaming.

With Splice Machine's native Spark DataSource, you can perform transactional database operations directly on DataFrames and receive DataFrames as ResultSets of arbitrary ANSI-SQL queries. This means that Splice Machine’s full transactional capabilities are available to developers without requiring them to change the way they do data engineering and data science. We'll see examples of this below.
How to Use the Splice Machine Native Spark DataSource

This topic will help you to get started with using the Splice Machine Native Spark DataSource (aka the Spark Adapter) in your applications and Zeppelin notebooks, in the following sections:

- **The SplicemachineContext Class**
- **Database Permissions and the Native Spark DataSource**
- **Accessing Database Objects with Internal Access**

See the Native Spark DataSource Overview topic for an overview of the Native Spark DataSource.

**The SplicemachineContext Class**

The first thing you need to do when using the Native Spark DataSource is to create an instance of the SplicemachineContext class; this is the primary serializable class that you can broadcast in your Spark applications. This class interacts with your Splice Machine cluster in your Spark executors, and provides the methods that you can use to perform operations such as:

- Interfacing with the Splice Machine RDD
- Running inserts, updates, and deletes on your data
- Converting data types between Splice Machine and Spark

**Creating Your Context**

Here's an example of creating a context in an interactive Spark Shell session:

```scala
import com.splicemachine.spark.splicemachine._
import com.splicemachine.derby.utils._
import com.splicemachine.derby.impl.SpliceSpark
SpliceSpark.setContext(sc)
val spliceJDBC = "jdbc:splice://SPLICESERVERHOST:1527/splicedb;user=<yourUserId>;password=<yourPassword>"
val SpliceContext = new SplicemachineContext(spliceJDBC)
val ds = SpliceContext.df("select * from splice.test")
```

And here's a similar example from a Zeppelin notebook:

```scala
%spark
import com.splicemachine.spark.splicemachine._
import com.splicemachine.derby.utils._

val JDBC_URL = "jdbc:splice://:1527/splicedb;user=<yourname>;password=<yourpswd>"
val splicemachineContext = new SplicemachineContext(JDBC_URL)
```
Using the Context to Populate a Table

Here's a simple example that illustrates how to populate a Splice Machine table from a Spark DataFrame. This example uses Zepplin.

First we'll create a DataFrame with a little data in it:

```
%spark
val carsDF = Seq(
  (1, "Toyota", "Camry"),
  (2, "Honda", "Accord"),
  (3, "Subaru", "Impreza"),
  (4, "Chevy", "Volt")
).toDF("NUMBER", "MAKE", "MODEL")
```

Then we'll create a table in our Splice Machine database:

```
%splicemachine
create table mySchema.carsTbl ( number int primary key,
     make  varchar(20),
     model varchar(20) );
```

Now we can insert the contents of our DataFrame directly into the table:

```
%spark
splicemachineContext.insert( carsDF, "mySchema.carsTbl" );
```

Though this simple example contains little data, you can operate on a DataFrame of any size in exactly the same way.

The Running Apps with the Native Spark DataSource and Using our Native Spark DataSource with Zeppelin topics in this chapter contains example walkthroughs that show you how to use the context object to interact with your Splice Machine database.

Accessing Database Objects with Internal Access

By default, Native Spark DataSource queries execute in the Spark application, which is highly performant and allows access to almost all Splice Machine features. However, when your Native Spark DataSource application uses our Access Control List (ACL) feature, there is a restriction with regard to checking permissions.

The specific problem is that the Native Spark DataSource does not have the ability to check permissions at the view level or column level; instead, it checks permissions on the base table. This means that if your Native Spark DataSource application doesn't have access to the table underlying a view or column, it will not have access to that view or column; as a result, a query against the view or column fails and throws an exception.
The workaround for this problem is to tell the Native Spark DataSource to use *internal* access to the database; this enables view/column permission checking, at a very slight cost in performance. With internal access, the adapter runs queries in Splice Machine and temporarily persists data in HDFS while running the query.

**NOTE:** The ACL feature is enabled by setting the property `splice.authentication.token.enabled = true`.

### Context Connection Options

When using the Native Spark DataSource, you can specify some optional properties for the JDBC connection that you’re using to access your Splice Machine database. To do so, Map those options using a `SpliceJDBCOptions` object, and then create your `SplicemachineContext` with that map. For example:

```scala
val options = Map(
    JDBCOptions.JDBC_URL -> "jdbc:splice://<jdbcUrlString>",
    SpliceJDBCOptions.JDBC_INTERNAL_QUERIES -> "true"
)
spliceContext  = new SplicemachineContext( options )
```

The `SpliceJDBCOptions` properties that you can currently specify in the JDBC connect URL are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC_INTERNAL_QUERIES</td>
<td>false</td>
<td>A string with value <code>true</code> or <code>false</code>, which indicates whether or not to run queries internally by default.</td>
</tr>
<tr>
<td>JDBC_TEMP_DIRECTORY</td>
<td>/tmp</td>
<td>The path to the temporary directory that you want to use when persisting temporary data from internally executed queries.</td>
</tr>
</tbody>
</table>

The user running a query must have **write permission** on this directory, or your connected application may freeze or fail.

### Database Permissions and the Native Spark DataSource

To use the Splice Native Spark DataSource, a user must have execute permission on the following four system procedures:

- `SYCS_HBASE_OPERATION`
These procedures are all Splice Machine system procedures that are used internally to efficiently perform direct HBASE and HDFS operations. They are not documented because they are intended only for use by the Splice Machine code itself; however, the Native Spark DataSource uses these procedures, so any user of the Adapter must have permission to execute them.

Here's an example of granting execute permission to these procedures to a user named myUserName:

```
splice> grant execute on procedure SYCS_UTIL.SYSCS_HBASE_OPERATION to myUserName;
0 rows inserted/updated/deleted
splice> grant execute on procedure SYCS_UTIL.SYSCS_HDFS_OPERATION to myUserName;
0 rows inserted/updated/deleted
splice> grant execute on procedure SYCS_UTIL.SYSCS_GET_SPLICE_TOKEN to myUserName;
0 rows inserted/updated/deleted
splice> grant execute on procedure SYCS_UTIL.SYSCS_CANCEL_SPLICE_TOKEN to myUserName;
0 rows inserted/updated/deleted
```

**Using the Native Spark DataSource with Kerberos**

If you're using the Native Spark DataSource on a Kerberized cluster, you must set the following property value in your hbase-site.xml settings file:

```
splice.authentication.token.enabled=true
```
The Native Spark DataSource Methods

This topic describes the following methods of the SplicemachineContext class:

- `analyzeSchema`
- `analyzeTable`
- `bulkImportHFile`
- `createTable`
- `delete`
- `df and internalDf`
- `dropTable`
- `export`
- `exportBinary`
- `getConnection`
- `getSchema`
- `insert`
- `rdd and internalRdd`
- `splitAndInsert`
- `tableExists`
- `truncateTable`
- `update`
- `upsert`

**analyzeSchema**

This method collects statistics for an entire schema; it is the same as using the `ANALYZE SCHEMA` splice> command line.

```scala
analyzeSchema(schemaName: String): Unit
```

*schemaName*

The name of the schema that you want analyzed.
**analyzeTable**

This method collects statistics for a specific table; it is the same as using the `ANALYZE TABLE` splice> command line.

```scala
analyzeTable( tableName: String,
                estimateStatistics: Boolean = false,
                samplePercent: Double = 0.10 ): Unit
```

- **tableName**
  The name of the table that you want analyzed.

- **estimateStatistics**
  A Boolean that specifies whether you want statistics generated by sampling the specified sampling percentage of the table. This can significantly reduce the overhead associated with generating statistics. Setting this parameter to `false` specifies that statistics are to be generated based on the full table.

- **samplePercent**
  A value between 0 and 100 that specifies the sampling percentage to use when generating statistics for this table. This value defaults to 10 percent, and is only used if `estimateStatistics` is set to `true`.

**bulkImportHFile**

This method efficiently imports data into your Splice Machine database by first generating HFiles and then importing those HFiles; it is the same as using the Splice Machine `SYSCS_UTIL.BULK_IMPORT_HFILE` system procedure.

You can either pass the data to this method in a DataFrame, or you can pass the data in an RDD, and pass in a structure (the Catalyst schema) that specifies the organization of the data.

```scala
bulkImportHFile( dataFrame: DataFrame,
                  schemaTableName: String,

bulkImportHFile( rdd: JavaRDD[Row],
                  schema: StructType,
                  schemaTableName: String,
```

- **dataFrame**
  The DataFrame containing the rows that you want imported into your database table.

- **schemaTableName**
  The combined schema and table names, in the form: `mySchema.myTable`.

- **rdd**
  The RDD containing the data the you want imported into your database table.

  *The Catalyst schema of the master table.*
  A structure that specifies the layout of the data in the RDD.
options
A collection of (key, value) pairs specifying the import options. For example, you can specify that sampling is not to be used with a statement like this:

```scala
val bulkImportOptions = scala.collection.mutable.Map( "skipSampling" -> "true" )
```

**createTable**
This method creates a new table in your Splice Machine database; it is the same as using the Splice Machine `CREATE TABLE` SQL statement.

```scala
createTable( tableName: String,
            structType: StructType,
            keys: Seq[String],
            createTableOptions: String ): Unit
```

*tableName*
The name of the table.

*structType*
A structure that specifies the table's schema.

*keys*
A sequence (comma-separated list) of keys for the table.

*createTableOptions*
A string that specifies the table options.

**delete**
This method deletes the contents of a Spark DataFrame or Spark RDD from a Splice Machine table; it is the same as using the Splice Machine `DELETE FROM` SQL statement.

You can either pass the data to this method in a DataFrame, or you can pass the data in an RDD, and pass in a structure that specifies the organization of the data.

```scala
delete( dataFrame: DataFrame,
         schemaTableName: String ): Unit
delete( rdd: JavaRDD[Row],
         schema: StructType,
         schemaTableName: String ): Unit
```

*dataFrame*
The DataFrame containing the rows that you want deleted from your database table.

*schemaTableName*
The combined schema and table names, in the form: mySchema.myTable.

rdd
The RDD containing the data the you want deleted from your database table.

Catalyst schema of the master table.
A structure that specifies the layout of the data in the RDD.

**df and internalDf**
These methods executes an SQL string within Splice Machine and returns the results in a Spark DataFrame.

```scala
df( sql: String ): Dataset[Row]
internalDf( sql: String ): Dataset[Row]
```

**sql**
The SQL string.

**dropTable**
This method removes the specified table; it is the same as using the Splice Machine `DROP TABLE` SQL statement.

You can pass the schema and table names into this method separately, or in combined (schema.table) format.

```scala
dropTable(schemaTableName: String): Unit
dropTable( schemaName: String, tableName: String ): Unit
```

**schemaTableName**
The combined schema and table names, in the form: mySchema.myTable.

**schemaName**
The schema name.

**tableName**
The name of the table.
**export**

This method exports a dataFrame in CSV format.

```scala
export( dataFrame: DataFrame,
        location: String,
        compression: Boolean,
        replicationCount: Int,
        fileEncoding: String,
        fieldSeparator: String,
        quoteCharacter: String): Unit
```

**dataFrame**

The dataFrame that you want to export.

**location**

The directory in which you want the export file(s) written.

**compression**

Whether or not to compress the exported files. You can specify one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The exported files are compressed using deflate/gzip.</td>
</tr>
<tr>
<td>false</td>
<td>Exported files are not compressed.</td>
</tr>
</tbody>
</table>

**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 (").
**exportBinary**

This method exports a dataFrame in binary format, generating one or more binary files, which are stored in the directory that you specify in the `location` parameter. More than one output file can be generated to enhance the parallelism and performance of this operation.

```scala
exportBinary( dataFrame: DataFrame,
             location: String,
             compression: Boolean,
             format: String): Unit
```

**dataFrame**
The dataFrame that you want to export.

**location**
The directory in which you want the export file(s) written.

**compression**
Whether or not to compress the exported files. You can specify one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The exported files are compressed using Snappy.</td>
</tr>
<tr>
<td>false</td>
<td>Exported files are not compressed.</td>
</tr>
</tbody>
</table>

If `compression=true`, then each of the generated files is named with this format: `part-r-<N>.snappy.parquet`; if not, then each file is named with this format: `part-r-<N>.parquet`. In either case, the value of `N` is a sequence of numbers and letters.

**format**
The format in which to write the exported file(s). The only format supported at this time is `parquet`.

**getConnection**

This method returns the current connection.

```scala
getConnection(): Connection
```

**getSchema**

This method returns the Catalyst schema of the specified table.
getSchema( schemaTableName: String ): StructType

*schemaTableName*
The combined schema and table names, in the form: `mySchema.myTable`.

**insert**
This method inserts the contents of a Spark DataFrame or Spark RDD into a Splice Machine table; it is the same as using the Splice Machine `INSERT INTO` SQL statement.

You can either pass the data to this method in a DataFrame, or you can pass the data in an RDD, and pass in a structure that specifies the organization of the data.

```
insert( dataFrame: DataFrame,
        schemaTableName: String ): Unit
```

```
insert( rdd: JavaRDD[Row],
        schema: StructType,
        schemaTableName: String ): Unit
```

*dataFrame*
The DataFrame containing the rows that you want inserted into your database table.

*schemaTableName*
The combined schema and table names, in the form: `mySchema.myTable`.

*rdd*
The RDD containing the data you want inserted into your database table.

*schema*
The Catalyst schema of the master table.

**rdd and internalRdd**
These methods create a Spark RDD from a Splice Machine table.

⚠️ The only difference between the `rdd` and `internalRdd` methods is that the `internalRdd` method runs internally and temporarily persists data on HDFS; this has a slight performance impact, but allows for checking permissions on Views. For more information, please see the Accessing Database Objects section in our *Using the Native Spark DataSource* topic.
rdd( schemaTable: String, 
columnProjection: Seq[String] = Nil ): RDD[Row]  

internalRdd( schemaTable: String, 
columnProjection: Seq[String] = Nil ): RDD[Row]  

**schemaTableName**  
The combined schema and table names, in the form: mySchema.myTable.

**columnProjection**  
The names of the columns in the underlying table that you want to project into the RDD; this is a comma-separated list of strings.

### splitAndInsert

This method improves the performance of inserting data from the Native DataSource: instead of inserting into a single HBase region and having HBase split that region, we pre-split the table based on the data we're inserting, and then insert the dataFrame. The table splits are computed by sampling the data in the dataFrame; Splice Machine uses the sampling percentage specified by the `sampleFraction` parameter value.

**splitAndInsert( dataFrame: DataFrame, 
schemaTableName: String, 
sampleFraction: Double): Unit**

**dataFrame**  
The dataFrame to sample.

**schemaTableName**  
The combined schema and table names, in the form: mySchema.myTable.

**sampleFraction**  
A value between 0 and 1 that specifies the percentage of data in the dataFrame that should be sampled to determine the splits. For example, specify 0.005 if you want 0.5% of the data sampled.

### tableExists

This method returns `true` if the specified table exists in your Splice Machine database.

You can pass the schema and table names into this method separately, or in combined (schema.table) format.

**tableExists( schemaTableName: String ): Boolean**

**tableExists( schemaName: String, 
tableName: String ): Boolean**

**schemaTableName**
The combined schema and table names, in the form: mySchema.myTable.

**schemaName**
The schema name.

**tableName**
The name of the table.

---

**truncateTable**
This method quickly removes all content from the specified table and returns it to its initial empty state.

It is the same as using the Splice Machine `TRUNCATE TABLE` SQL statement.

```scala
truncateTable( tableName: String ): Unit
```

**tableName**
The name of the table.

---

**update**
This method updates a Splice Machine table using the contents of a Spark DataFrame or Spark RDD from a Splice Machine table; it is the same as using the Splice Machine `DELETE FROM` SQL statement.

You can either pass the data to this method in a DataFrame, or you can pass the data in an RDD, and pass in a structure that specifies the organization of the data.

```scala
update( dataFrame: DataFrame,
        schemaTableName: String ): Unit

update( rdd: JavaRDD[Row],
        schema: StructType,
        schemaTableName: String ): Unit
```

**dataFrame**
The DataFrame containing the rows that you want updated in your database table.

**schemaTableName**
The combined schema and table names, in the form: mySchema.myTable.

**rdd**
The RDD containing the data the you want updated in your database table.

**schema**
The Catalyst schema of the master table.
**upsert**

This method upserts (inserts new records and updates existing records) the contents of a Spark DataFrame or Spark RDD into a Splice Machine table; it is the same as using the Splice Machine `SYSUTIL.UPSERT_DATA_FROM_FILE` system procedure.

You can either pass the data to this method in a DataFrame, or you can pass the data in an RDD, and pass in a structure that specifies the organization of the data.

```scala
upsert(dataFrame: DataFrame,
       schemaTableName: String): Unit

upsert(rdd: JavaRDD[Row],
       schema: StructType,
       schemaTableName: String): Unit
```

- **dataFrame**
  The DataFrame containing the rows that you want inserted into your database table.

- **schemaTableName**
  The combined schema and table names, in the form: `mySchema.myTable`.

- **rdd**
  The RDD containing the data the you want inserted into your database table.

- **schema**
  The Catalyst schema of the master table.

**See Also**

- Using the Splice Machine Native Spark DataSource
- Using Spark Submit
- Using our Native Spark DataSource with Zeppelin
Running Apps with the Native Spark DataSource

This topic shows you how to submit a job to the Splice Machine Native Spark DataSource in two ways:

- Using the Native Spark DataSource Interactively with Spark Shell
- Submitting an App With Spark Submit

Using the Native Spark DataSource Interactively with Spark Shell

This section provides examples of using the interactive Spark Shell with the Splice Machine Native Spark DataSource, in these subsections:

- Using Spark Shell on a HDP cluster
- Using Spark Shell on a CDH cluster
- Example Program

Using Spark Shell on a HDP cluster

This command starts the Spark Shell on a Hortonworks cluster for use with the Splice Machine Native DataSource:

```
spark-shell \
--conf "spark.dynamicAllocation.enabled=false" \
--conf "spark.task.maxFailures=2" \
--conf "spark.driver.memory=4g" \
--conf "spark.driver.cores=1" \
--conf "spark.kryoserializer.buffer=4m" \
--conf "spark.kryoserializer.buffer.max=100m" \
--conf "spark.kryo.registrator=com.splicemachine.derby.impl.SpliceSparkKryoRegistrator" \
--conf "spark.io.compression.codec=org.apache.spark.io.SnappyCompressionCodec" \
--conf "spark.executor.extraClassPath=/etc/hbase/2.6.3.0-235/0:/var/lib/splicemachine/*:/usr/hdp/2.6.3.0-235/jars/*:/usr/hdp/2.6.3.0-235/hbase/lib/*" \
--conf "spark.driver.extraClassPath=/etc/hbase/2.6.3.0-235/0:/var/lib/splicemachine/*:/usr/hdp/2.6.3.0-235/jars/*:/usr/hdp/2.6.3.0-235/hbase/lib/*" \
--conf "spark.kryo.registrator=com.splicemachine.derby.impl.SpliceSparkKryoRegistrator" \
--jars "/var/lib/splicemachine/splicemachine-hdp2.6.3-2.2.0.2.6.3.0-235_2.11-2.7.0.1836-SNAPSHOT.jar" \
--master yarn
```

Using Spark Shell on a CDH cluster

This command starts the Spark Shell on a Cloudera cluster for use with the Splice Machine Native DataSource:
Example Program
Here's a example of an interactive Spark Shell session that uses our Native DataSource to access a Splice Machine database:

```scala
import com.splicemachine.spark.splicemachine._
import com.splicemachine.derby.utils._
import com.splicemachine.derby.impl.SpliceSpark
SpliceSpark.setContext(sc)
val spliceJDBC = "jdbc:splice://SPLICESERVERHOST:1527/spicedb;user=<yourUserId>;password=<yourPassword>"
val SpliceContext = new SplicemachineContext(spliceJDBC)
val ds = SpliceContext.df("select * from splice.test")
ds.count
ds.show
```

Submitting an App With Spark Submit
You can run your compiled apps with the Native Spark DataSource with our spark-submit.sh script.

Configuring the spark-submit.sh Script
You need to change some of the values at the top of this script to configure it for your environment and your app; these values typically need to be modified:

```
spark2-shell
--conf "spark.dynamicAllocation.enabled=false"
--conf "spark.task.maxFailures=2"
--conf "spark.driver.memory=4g"
--conf "spark.driver.cores=1"
--conf "spark.kryoserializer.buffer=4m"
--conf "spark.kryoserializer.buffer.max=100m"
--conf "spark.kryo.registrator=com.splicemachine.derby.impl.SpliceSparkKryoRegistrar"
--conf "spark.io.compression.codec=org.apache.spark.io.SnappyCompressionCodec"
--conf "spark.driver.extraClassPath=/opt/cloudera/parcels/CDH/lib/hbase/conf:/opt/cloudera/parcels/SPLICEMACHINE/lib/*:/opt/cloudera/parcels/SPARK2/lib/spark2/jars/*:/opt/cloudera/parcels/CDH/lib/hbase/lib/*"
--conf "spark.kryo.registrator=com.splicemachine.derby.impl.SpliceSparkKryoRegistrar"
--jars "./sparksplice.jar"
--master yarn
```
TargetTable=<tableName>
TargetSchema=<schemaName>
RSHostName=<serverId, e.g. localhost>
SpliceConnectPort=<portId>
UserName=<yourUserName>
UserPassword=<yourPassword>
KafkaBroker=<Kafka Broker ID>
KafkaTopic=<Registered Kafka Topic ID>
KrbPrincipal=<Kerberos Principal>
KrbKeytab=<Kerberos Keytab Location>

The spark-submit.sh Script
Here's an example of the spark-submit.sh script
#!/bin/bash
export SPARK_KAFKA_VERSION=0.10

TargetTable=TEST_TABLE
TargetSchema=SPLICE
RSHostName=localhost
SpliceConnectPort=1527
UserName=splice
UserPassword=admin
KafkaBroker=stl-colo-srv136
KafkaTopic=test-k
KrbPrincipal=hbase/stl-colo-srv136.splicemachine.colo@SPLICEMACHINE.COLO
KrbKeytab=/tmp/hbase.keytab

spark2-submit --conf "spark.driver.extraJavaOptions=-Dsplice.spark.yarn.principal=hbase/stl-colo-srv136.splicemachine.colo \
-Dsplice.spark.yarn.keytab=/tmp/hbase.keytab \
-Dsplice.spark.enabled=true \
-Dsplice.spark.app.name=SpliceETLApp \
-Dsplice.spark.master=yarn-client \
-Dsplice.spark.logConf=true \
-Dsplice.spark.yarn.maxAppAttempts=1 \
-Dsplice.spark.driver.maxResultSize=3g \
-Dsplice.spark.driver.cores=4 \
-Dsplice.spark.yarn.am.memory=2g \
-Dsplice.spark.dynamicAllocation.enabled=true \
-Dsplice.spark.dynamicAllocation.executorIdleTimeout=30 \
-Dsplice.spark.dynamicAllocation.cachedExecutorIdleTimeout=30 \
-Dsplice.spark.dynamicAllocation.minExecutors=8 \
-Dsplice.spark.dynamicAllocation.maxExecutors=17 \
-Dsplice.spark.memory.fraction=0.6 \
-Dsplice.spark.scheduler.mode=FAIR \
-Dsplice.spark.serializer=org.apache.spark.serializer.KryoSerializer \
-Dsplice.spark.shuffle.service.enabled=true \
-Dsplice.spark.yarn.am.extraLibraryPath=/opt/cloudera/parcels/CDH/lib/hadoop/lib/native \
-Dsplice.spark.driver.extraJavaOptions=-Dlog4j.configuration=file:/etc/spark/conf/log4j.properties \
-Dsplice.spark.driver.extraLibraryPath=/opt/cloudera/parcels/CDH/lib/hadoop/lib/native \
-Dsplice.spark.driver.extraClassPath=/opt/cloudera/parcels/CDH/lib/hbase/conf:/opt/cloudera/parcels/CDH/jars/htrace-core-3.2.0-incubating.jar \
-Dsplice.spark.executor.extraLibraryPath=/opt/cloudera/parcels/CDH/lib/hadoop/lib/native \
-Dsplice.spark.executor.extraClassPath=/opt/cloudera/parcels/CDH/lib/hbase/conf:/opt/cloudera/parcels/CDH/jars/htrace-core-3.2.0-incubating.jar \
-Dsplice.spark.eventLog.enabled=true \
-Dsplice.spark.eventLog.dir=hdfs:///user/spark/spark2ApplicationHistory \
-Dsplice.spark.local.dir=/tmp \
-Dsplice.spark.yarn.jars=/opt/cloudera/parcels/SPLICEMACHINE/lib/* \
-Dsplice.spark.ui.port=4042" \
--conf "spark.dynamicAllocation.enabled=false"
--conf "spark.streaming.stopGracefullyOnShutdown=true" \
--conf "spark.streaming.kafka.maxRatePerPartition=500" \
--conf "spark.streaming.kafka.consumer.cache.enabled=false" \
--conf "spark.streaming.concurrentJobs=1" \
--conf "spark.task.maxFailures=2" \
--conf "spark.driver.memory=4g" \
--conf "spark.driver.cores=1" \
--conf "spark.kryoserializer.buffer=1024" \
--conf "spark.kryoserializer.buffer.max=2047" \
--conf "spark.io.compression.codec=org.apache.spark.io.SnappyCompressionCodec" \
--conf "spark.executor.extraClassPath=/etc/hadoop/conf:/etc/hbase/conf:/opt/cloudera/parcels/SPlicEMACHINE/lib/*:/opt/cloudera/parcels/SPARK2/lib/spark2/jars/*:/opt/cloudera/parcels/CDH/lib/hbase/*" \
--conf "spark.driver.extraClassPath=/etc/hadoop/conf:/etc/hbase/conf:/opt/cloudera/parcels/SPlicEMACHINE/lib/*:/opt/cloudera/parcels/SPARK2/lib/spark2/jars/*:/opt/cloudera/parcels/CDH/lib/hbase/*" \
--files "/etc/spark/conf/log4j.properties,/etc/krb5.conf" \
--keytab "/tmp/hbase.keytab" \
--principal "hbase/stl-colo-srv136.splicemachine.colo" \
--name "DataGen" \
--jars "splicemachine-cdh5.8.3-2.1.0-2.11-2.5.0.1803-SNAPSHOT.jar,spark-streaming-kafka-0-10_2.11-2.2.0.cloudera1.jar" \
--class com.splice.custom.reader.Main \
--master yarn --deploy-mode cluster --num-executors 4 --executor-memory 10G --executor-cores 1 /home/splice/stream-app/target/reader-1.0-SNAPSHOT.jar \
$TargetTable $TargetSchema $RSHostName $SpliceConnectPort $UserName $UserPassword $KafkaBroker $KafkaTopic

See Also

- Using the Native Spark DataSource
- Native Spark DataSource Methods
- Using Our Native Spark DataSource with Zeppelin
Using the Native Spark DataSource in Zeppelin Notebooks

This topic shows you how to use the Native Spark DataSource in an Apache Zeppelin notebook. We use the `%spark` and `%splicemachine` Zeppelin interpreters to create a simple Splice Machine database table, and then access and modify that table, in these steps:

- 1. Set up the Native Spark DataSource
- 2. Create a Table in Your Database
- 3. Populate the Table
- 4. Perform Table Operations

We have posted a [blog article](#) on our website walks that you through this Zeppelin notebook example in greater detail.

1. Set Up the Native Spark DataSource

Before you can use the Native Spark DataSource, you need to create a `SplicemachineContext` object, specifying the URL to use to connect to your database. For example:

```scala
%spark
import com.splicemachine.spark.splicemachine._
import com.splicemachine.derby.utils._
val JDBC_URL = "jdbc:splice://XXXX:1527/splicedb;user=YourUserId;password=YourPassword"
val SpliceContext = new SplicemachineContext(JDBC_URL)
```

The Native Spark DataSource has a few special (optional) requirements related to database permissions, which you can configure in your JDBC connection URL; for information, please see the Accessing Database Objects section in the *Using the Native Spark DataSource* topic in this chapter.

2. Create and Populate a Table in Your Database

Now let's create a simple table in our Splice Machine database:

```splicemachine
%splicemachine
drop table if exists carsTbl;
create table carsTbl ( number int primary key, make varchar(20), model varchar(20) );
```

3. Use a DataFrame to Populate the Table

Next we'll create and populate a Spark DataFrame with some data:
%spark
val carsDF = Seq(
   (1, "Toyota", "Camry"),
   (2, "Honda", "Accord"),
   (3, "Subaru", "Impreza"),
   (4, "Chevy", "Volt")
).toDF("NUMBER", "MAKE", "MODEL")

Then we use the Splice Machine Native Spark DataSource to insert that data into our database table:

%spark
SpliceContext.insert(carsDF, "SPLICE.CARSTBL")

4. Perform Table Operations
Now we can use the Native Spark DataSource to directly interact with our database table using Spark, as shown in the following basic table operations examples:

- Selecting Data From the Table
- Updating Data In the Table
- Deleting Data From the Table
- Dropping the Table

NOTE: This section provides simple examples of using the Native Spark DataSource to perform several simple database operations; for a complete list of operations available from the DataSource, see the Native Spark DataSource API topic in this chapter.

Select Data from the Table
You can use Spark with the Adapter to select data from your table just as you would with the splice> command line interface:

%spark
SpliceContext.df("SELECT * FROM SPLICE.CARSTBL").show()

Update Data in the Table
You can use Spark with the Adapter to update data in your table just as you would with the splice> command line interface:
Delete Data From the Table
You can also use Spark with the Adapter to delete data from your table just as you would with the splice> command line interface:

```scala
%spark
val deleteCarsDF = Seq(
  (1, "Toyota", "Rav 4 XLE"),
  (4, "Honda", "Accord Hybrid")
).toDF("NUMBER", "MAKE", "MODEL")
SpliceContext.delete(deleteCarsDF, "SPLICE.CARSTBL")
```

Drop the Table
And you can use Spark with the Adapter to drop your table just as you would with the splice> command line interface:

```scala
%spark
if (SpliceContext.tableExists("SPLICE.CARSTBL")) {
  SpliceContext.dropTable("SPLICE.CARSTBL")
}
```

See Also

- Native Spark DataSource Methods
- Using Spark Submit
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 SPLICE_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. DEVELOPER QUICK START → Download our JDBC, ODBC, and command line (sqlshell) clients to easily connect to your database.
4. 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.
5. 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
- An introduction to Using Zeppelin with Splice Machine

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.

**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:

<table>
<thead>
<tr>
<th>Support Type</th>
<th>Pricing</th>
<th>Support Feature</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard Support</strong></td>
<td>Free</td>
<td>Coverage Hours</td>
<td>Monday-Friday 9am-6pm Pacific time</td>
<td>(subject to local holidays)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>System Impaired</td>
<td>Significant issues with speed, quality, or functionality of Service.</td>
<td>&lt; 12 business hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other Issues</td>
<td>General queries and guidance requests</td>
<td>&lt; 24 business hours</td>
</tr>
<tr>
<td><strong>Business Support</strong></td>
<td>As per contract includes SLA</td>
<td>Coverage Hours</td>
<td>24 hours a day, 7 days a week, 365 days a year</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Production System Down</td>
<td>Complete loss of Service on Production cluster.</td>
<td>&lt; 1 hour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Production System Impaired</td>
<td>Significant issues with speed, quality, or functionality of Service on Production cluster.</td>
<td>&lt; 4 hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Production System Down</td>
<td>Significant issues with speed, quality, or functionality of Service on non-production cluster.</td>
<td>&lt; 12 hours</td>
</tr>
</tbody>
</table>
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

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:

<table>
<thead>
<tr>
<th>UI Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dashboard</td>
<td>The Splice Machine Dashboard or <strong>Cloud Manager</strong> 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.</td>
</tr>
<tr>
<td>Cluster Manager</td>
<td>Use the Cluster Manager to create new clusters and to monitor the health of your clusters.</td>
</tr>
<tr>
<td>Notebooks Manager</td>
<td>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.</td>
</tr>
<tr>
<td>Database Console</td>
<td>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.</td>
</tr>
<tr>
<td>Events Manager</td>
<td>Our <strong>Events Manager</strong> allows you to examine events that have occurred on your cluster.</td>
</tr>
<tr>
<td>Account Manager</td>
<td>The Splice Machine Account Manager is where you manage your users, your profile, and your billing information.</td>
</tr>
</tbody>
</table>
# Splice Machine Cloud Manager

This guide helps you to get registered with and start using the Splice Machine Cloud Manager. Here are the topics included in this guide:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Navigating Your Dashboard</strong></td>
<td>Your <strong>Dashboard</strong> 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.</td>
</tr>
<tr>
<td><strong>Registering</strong></td>
<td>Shows you how to complete the first step of using your database service: registering as a user of the Splice Machine Cloud Manager.</td>
</tr>
<tr>
<td><strong>Logging In</strong></td>
<td>Shows you how to log into the Splice Machine Cloud Manager once you've registered.</td>
</tr>
<tr>
<td><strong>Creating a New Cluster</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
| **Best Practices: Ingesting Data** | You can load your data into Splice Machine from an AWS S3 bucket:  
  - 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 Best Practices: Ingesting Data topic to load that data into Splice Machine.  
  Also note that the S3 directory you specify for the log of record import issues (the **bad record** 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. |
<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the DB Console</td>
<td>The <strong>Splice Machine Database Console</strong> 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. 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 <strong>DB Console</strong> link in your Cluster Management dashboard, or by following the link sent to you by Splice Machine when your cluster was originally created.</td>
</tr>
</tbody>
</table>
| Using the Notebooks Manager | 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. This tutorial walks you through using a notebook created by Splice Machine that:  
  - 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 |
| Managing Your Account | Use our Account Manager to manage your company profile, billing, and users.                                                                                                                                 |
| Reviewing Event Notifications | 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:

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.

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 and Options** 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:
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:
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.

For example, here are the pop-ups from the Create New Cluster screen:
Click the information button again to dismiss a pop-up.

**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.
<table>
<thead>
<tr>
<th><strong>Cluster Name</strong></th>
<th>Supply whatever name you want for your Splice Machine cluster.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cloud Provider</strong></td>
<td>You can select which cloud provider is hosting your cluster by clicking the current provider name, which drops down a list of choices.</td>
</tr>
<tr>
<td><strong>NOTE:</strong> If you have subscribed to Splice Machine via the AWS Marketplace, your costs will be estimated on an hourly basis instead of on a monthly basis.</td>
<td></td>
</tr>
<tr>
<td><strong>Region</strong></td>
<td>You can select in which region your cluster will reside by clicking the current region name, which drops down a list of choices.</td>
</tr>
<tr>
<td><strong>Data Sizing</strong></td>
<td>Move the slider to modify your estimate of how large your database will be.</td>
</tr>
<tr>
<td><strong>Internal Dataset (TB)</strong></td>
<td>Internal Dataset is the amount of data that you will be storing within your Splice Machine database.</td>
</tr>
<tr>
<td><strong>Dedicated Storage</strong></td>
<td>Select this checkbox to have us provision dedicated storage for your database instance, which does add cost. Leave this unselected to have your database instance stored on shared hardware.</td>
</tr>
<tr>
<td><strong>External Dataset (TB)</strong></td>
<td>Move the slider to modify your estimate of how large your external dataset will be. External Dataset is the amount of data you will be accessing from external data sources, using features such as external tables and our virtual table interface.</td>
</tr>
<tr>
<td><strong>Cluster Power</strong></td>
<td>Move the slider to modify your estimate of how much processing power you need for transactional activity, involving quick inserts, lookups, updates, and deletes. More OLTP units means more region servers in your cluster.</td>
</tr>
<tr>
<td><strong>OLTP Splice Units</strong></td>
<td>Move the slider to modify your estimate of how much processing power you need for running longer queries, typically analytical queries. More OLAP units means more Spark executors.</td>
</tr>
<tr>
<td><strong>OLAP Splice Units</strong></td>
<td>Move the slider to modify your estimate of how many Spark units should be utilized by the Splice Machine Native Spark Datasource and other external uses of Spark libraries, such as MLlib.</td>
</tr>
</tbody>
</table>
Enable

Select this checkbox to enable the Splice Machine ML Manager, which provides access to our Model Workflow and Deployment integration and additional Machine Learning libraries.

**Backup Frequency**

Frequency

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:

**Hourly:**

![Backup Frequency Hourly](image)

**Daily:**

![Backup Frequency Daily](image)

**Weekly:**

![Backup Frequency Weekly](image)

---

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:
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 and Options**

Once you've configured your cluster, click the Next button to display the Cluster Access and Options screen. The following image includes displays of the pop-up help information displays for the different access methods:
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 Access](https://aws.amazon.com/vpc/)* tutorial.

For more information about Amazon VPC, see [https://aws.amazon.com/vpc/](https://aws.amazon.com/vpc/).
For more information about Amazon IAM, see [https://aws.amazon.com/iam/](https://aws.amazon.com/iam/).

You can change the number of Zeppelin instances available on your cluster, and you can adjust how much Java memory is allocated for the Spark Interpreter in each instance. Multiple Zeppelin instances allow multiple users to develop and run notebooks independently.

You can also add (at an additional cost) our **Machine Learning Manager** to your cluster by clicking the **Enable button** in the **ML Manager** section at the bottom of this screen. The Splice Machine ML Manager facilitates machine learning development by integrating MLflow, Amazon Sagemaker deployment, additional Machine Learning libraries, and our database together.

After setting up any options and 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:

![Payment Method Options](image-url)
### Credit Card

<table>
<thead>
<tr>
<th>Credit Card</th>
<th>ACH</th>
<th>Authorization Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name on Card</td>
<td></td>
<td></td>
</tr>
<tr>
<td>My Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Credit Card Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4242 5678 9012 3456</td>
<td>09 / 17</td>
<td>999</td>
</tr>
<tr>
<td>Cancel</td>
<td>Submit</td>
<td></td>
</tr>
</tbody>
</table>

### ACH Electronic Transfer

<table>
<thead>
<tr>
<th>Credit Card</th>
<th>ACH</th>
<th>Authorization Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>TODO: Implement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cancel</td>
<td>Submit</td>
<td></td>
</tr>
</tbody>
</table>

### Authorization Code

<table>
<thead>
<tr>
<th>Credit Card</th>
<th>ACH</th>
<th>Authorization Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization Code:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3345-8888-9078-ABCF-4321</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cancel</td>
<td>Submit</td>
<td></td>
</tr>
</tbody>
</table>

### 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:
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 Ingestion Best Practices topic 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:

- Viewing and Managing Your Clusters
- Creating a New Cluster

Viewing and Managing Your Clusters

Your dashboard displays a list of the clusters that you've created, along with the status of each:

<table>
<thead>
<tr>
<th>Cluster Name</th>
<th>OLTP (Splice units)</th>
<th>OLAP (Splice units)</th>
<th>External (Spark units)</th>
<th>Storage (TB)</th>
<th>Creation Date</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>test1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0.1</td>
<td>2/1/2019 - 18:57</td>
<td>Deleted</td>
</tr>
<tr>
<td>test2</td>
<td>4</td>
<td>4</td>
<td>--</td>
<td>0.1</td>
<td>2/4/2019 - 19:47</td>
<td>Deleted</td>
</tr>
<tr>
<td>test3</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4.9</td>
<td>2/5/2019 - 21:30</td>
<td>Deleted</td>
</tr>
<tr>
<td>performance</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4.8</td>
<td>2/6/2019 - 15:40</td>
<td>Deleted</td>
</tr>
<tr>
<td>perf2</td>
<td>4</td>
<td>16</td>
<td>1</td>
<td>4.8</td>
<td>2/6/2019 - 21:10</td>
<td>Deleted</td>
</tr>
<tr>
<td>mlmanager</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>0.1</td>
<td>2/12/2019 - 20:05</td>
<td>Active</td>
</tr>
</tbody>
</table>

CPU, Memory, and Disk usage statistics are displayed for the currently selected cluster (in this case mlmanager). Simply click or tap a cluster name to display its usage information.
Creating a New Cluster

When you first log in to your Splice Machine Database Service, you'll see a large indication that you need to create a new cluster before you can do anything:

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. After you've created a new cluster, you land back on this Dashboard screen, at which time the status of your new cluster will be **Initializing**.

After a few minutes, your new cluster will be initialized, its status will change to **Active**, and you'll 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, as described in the Managing a Cluster topic.
Managing a Cluster

This topic describes the actions you can initiate from 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. You can use the controls in this screen to:

- Launch Tools for Monitoring and Notebook Development
- Modify Your Cluster
- View Your Cluster's Resource Usage
- View Cluster Events
- View Log Details for Your Cluster
- Download Drivers and Related Software

Launch Tools for Monitoring and Notebook Development

The Cluster Management screen includes three buttons that launch monitoring and development tools:

- Click the DB Console button to open a new browser tab for the Database Console, which you can use to monitor the Splice Machine queries running on your cluster in real-time. The DB Console uses port 4040.
- Click the Notebooks button to open a new browser tab for your Zeppelin notebooks. You can use the notebooks that are pre-installed on your cluster to learn about and experiment with using various features of Splice Machine; you can also quickly develop your own notebooks that use your database, Spark, and Machine Learning.
- Click the Spark Console button to open a new browser tab for the Spark Console, which you can use to monitor the Spark jobs running on your cluster in real-time. The Spark Console uses port 4041.

Modify Your Cluster

The Cluster Management has four buttons you can click to update your cluster:

- Click the Reconfigure button to reconfigure this cluster. This displays the cluster reconfiguration screen:
Reconfigure Cluster mlmanager

VPC Setup

- Client VPC connectivity required

  - Account ID:
  - VPC ID (Accepter):

IAM S3 Access

- S3 connectivity

  - S3 Access Key:
  - S3 Secret Key:

Zeppelin Setup

- Number of Instances: 2
- Spark Interpreter Java: 5120
  - Memory

Cancel  Reconfigure
You can modify your VPC and/or IAM configuration information or change the Zeppelin instances settings. Click the **Reconfigure** button to update your configuration and return to your Cluster Management screen.

- Click the **Resize** button to modify the size of this cluster. This displays the cluster resize screen, which is almost identical to the **Create New Cluster** screen:

![Cluster Resize Screen]

After you make any modifications to the cluster configuration, click the **Resize** button to update the cluster settings and return to your Cluster Management screen. Note that resizing a cluster can take a bit of time.

- Click the **Delete** button to delete this cluster. This displays a confirmation screen:
After confirming that you want to do so, your cluster will be deleted, and its status in your dashboard will show as Deleted.

Click the Retrieve DB Password button to display the user ID and password for the Splice Machine database running on this cluster.

View Your Cluster’s Resource Usage
The Activity tab displays timeline graphs of cluster CPU, Memory, and Storage usage. For example:
View Cluster Events

The *Events* tab displays information about cluster events that have occurred over a specified period of time:

As you can see, you're able to search for specific words in an event in addition to filtering on a range of dates.

View Log Details for Your Cluster

The *Events* tab displays information about cluster events that have occurred over a specified period of time:
As you can see, you’re able to search for specific words in a log entry in addition to restricting the display to a range of date/time values. You can also filter on any combination of log entry types (Info, Debug, Warning, and Error).

**Download Drivers and Related Software**

The bottom of the **Cluster Management** screen includes a number of useful links:

- **External Spark Console**
- **Download ODBC Driver**
- **Download JDBC Driver**
- **Download SqlShell command line client.**
- **Copy a basic JDBC connection URL to your cluster’s database.**

You can:

- Launch the **Spark Console** in a separate browser tab.
- Download the Splice Machine **ODBC Driver**.
- Download the Splice Machine **JDBC Driver**.
- Download the Splice Machine **SqlShell** command line client.
- Copy a basic JDBC connection URL to your cluster’s database.

The same JDBC URL 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 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
- Managing Users
- Reviewing and Updating Your Company Information

This is a DB-Service-Only topic! Learn more

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.

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 Account Drop-down:

The **Profile** screen displays:

![Profile Screen](image)

You can edit your profile information by clicking the **EDIT** button in the Profile Info panel.

**Managing 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 (we have redacted names and email addresses):
Adding a User via Invitation

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.
Removing a User

To remove yourself as a user, or to remove a user whom you invited to join, you can click the small trash can icon that displays on the right side of the listing for that user. The trash can icon only displays for users who you are eligible to remove. You’ll be asked to confirm the removal:

Rules for Adding, Modifying, and Removing Users

Splice Machine enforces the following rules regarding users in your account:

- Each Cloud Account must have at least 1 Primary user at all time.
A non-primary user can only delete him/herself.

A Primary user can delete non-primary users at any time.

You must be a Primary user to designate someone else as a Primary user.

You must be a Primary user to remove another user’s Primary status.

Only a Primary user can delete another Primary user.

A Primary user can only delete him/herself if there is another Primary user.

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:

![Company Information Screen]

To edit the company information associated click the Edit button.
Managing Your Event Notifications

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:

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.
Enabling Azure Active Directory Integration

To enable Azure Active Directory Integration for your Cloud-Managed Splice Machine service, you must create an Azure Application Registration.

An App Registration allows an application (web page, service, etc) to integrate with Azure Active Directory (AD) with a single sign-on. Having the application in Azure AD gives you control over the authentication policy for the application.

Follow these steps to create an Azure App Registration:

1. Log into Azure.
2. Select Azure Active Directory in the left sidebar.
3. Click App Registrations in the panel that displays:

4. Click New Registration in the menu at the top of the panel. This displays the Register an Application screen:
5. Fill in the following fields:

a. Enter the name you want to use to identify your application.
b. Select the supported account types; in most cases, you should select the **Accounts in this organizational directory only** option.
c. Enter the URLs that will use this App Registration in the **Redirect URI** section. Note that you can use the same APP Registration for multiple applications/environment. For example, if you are integrating with the Splice Machine Cloud Manager, your URLs would look something like this:

   - http://localhost:3000
   - https://cloud.splicemachine-qa.io/login
d. Click the *Register* button at the bottom of the screen to create your App Registration.

6. The **Overview** screen for your Application displays. **Make a note** of the Application (client) ID on this screen; you'll need this ID for integration.

7. Click the **Managed Application** link:

8. Now click **Permissions** in the left sidebar:
9. Click *Grant Admin consent for xxx*.
10. Select the directory and grant permissions.
11. Go back to the Application and select *Authentication* in the left sidebar to display the Authentication screen:
12. Scroll down in the center pane. Select these checkboxes in the *Advanced Settings* section:

- Access Tokens
- ID Tokens

13. Save your settings.
# Using Zeppelin Notebooks

This guide helps you to get started with using Zeppelin notebooks to interact with your Splice Machine database service.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Started with Zeppelin</td>
<td>Introduces you to using Zeppelin with your Splice Machine database.</td>
</tr>
<tr>
<td>Zeppelin Usage Notes</td>
<td>Specific usage notes for creating Zeppelin notebooks to use with Splice Machine.</td>
</tr>
<tr>
<td>A Simple Tutorial</td>
<td>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.</td>
</tr>
</tbody>
</table>
Getting Started with Zeppelin

This topic helps you to get started with using Zeppelin with your Splice Machine database service, in the following sections:

- The Zeppelin Dashboard
- Adding Your Credentials
- The Zeppelin Note Toolbar
- The Zeppelin Drop-Down Menu
- Monitoring Job Status
- Creating Notebooks
- Learn more

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.

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.
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 Simple Example tutorial.

### Adding Your Credentials

You use the Splice Machine interpreter (`%splicemachine`) in Zeppelin notebooks to interact with your Splice Machine database; this interpreter uses a JDBC connection to the database, and making that connection requires you to supply user credentials. Here’s how you can create the credentials to use with the Splice Machine interpreter in your Zeppelin notebooks:

1. Log in to Zeppelin, using the **Notebook** button, as described above.
2. Click the **Zeppelin** dropdown in the upper right corner of the window, and select **Credential**: 

The **Zeppelin Credentials Management** page displays:
3. Click the **Add** button to add your credentials:

![Credentials](image)

Enter **jdbc.splicemachine** as the **Entity**.

Use the same **Username** and **password** that you use to log into your database.

4. Click the **Save** button to add your credentials.

Now, when you specify the `%splicemachine` interpreter for a Zeppelin paragraph, your credentials will be used to connect to your Splice Machine database.

Remember to explicitly specify the `%splicemachine` interpreter in paragraphs, even if `%splicemachine` is the default interpreter for the notebook you're working on.

**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:
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:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Button" /></td>
<td>Executes all of the paragraphs in the note, in display-order sequence.</td>
</tr>
<tr>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>Shows or hides the code sections of the paragraphs in the note.</td>
<td></td>
</tr>
<tr>
<td>Shows or hides the result sections of the paragraphs in the note.</td>
<td></td>
</tr>
<tr>
<td>Clears the result sections of the paragraphs in the note.</td>
<td></td>
</tr>
<tr>
<td>Clones the current note.</td>
<td></td>
</tr>
<tr>
<td>Exports the current note in JSON format. <strong>NOTE:</strong> The code and result sections of all paragraphs are exported; you might want to clear your results before exporting a note.</td>
<td></td>
</tr>
<tr>
<td>Switches between personal and collaboration modes.</td>
<td></td>
</tr>
<tr>
<td>Commits changes that you've made to the content of the current note (and allows you to add a commit note).</td>
<td></td>
</tr>
<tr>
<td>Displays the revision you're currently viewing, and lets you select from available revisions.</td>
<td></td>
</tr>
<tr>
<td>Deletes the note.</td>
<td></td>
</tr>
<tr>
<td>Schedules execution of the note, using CRON syntax.</td>
<td></td>
</tr>
</tbody>
</table>

**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:
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.

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.

**NOTE:** We strongly encourage you to visit the [Zeppelin documentation site](#) to learn about creating, modifying, and running your own Zeppelin notebooks.
Zeppelin Usage Notes

This page currently contains exactly one tip about using Zeppelin with the Splice Machine database service; 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:

```scala
%spark
import java.util.Date
import java.sql.
{Connection, Timestamp}
classOfStringTimestamp
classOfObject
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:
```scala
%spark
import java.util.Date
import java.sql.
{Connection, Timestamp}
classOfTimestamp
classOfoo
val tt = Timestamp.valueOf("2261-12-31 00:00:00")
class foo extends Object { val xx: java.sql.Timestamp = java.sql.Timestamp.valueOf("2261-12-31 00:00:00") }

import java.util.Date
import java.sql.{Connection, Timestamp}
res14: Class java.sql.Timestamp = class java.sql.Timestamp
res15: Classfoo = class foo
tt: java.sql.Timestamp = 2261-12-31 00:00:00.0
defined class foo
FINISHED
```
A Simple Zeppelin Tutorial

This topic walks you through using a very simple Zeppelin notebook, to help you learn about using Zeppelin with the Splice Machine database service.

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:
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:
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:

Create a schema called EXAMPLE

```sql
CREATE SCHEMA EXAMPLE;
```

Query executed successfully. Affected rows: 0
Touched 4 sec. Last updated by splice at September 14, 2017, 10:09:30 AM.

Create a table called LINEITEM

```sql
CREATE TABLE EXAMPLE.LINEITEM (L_ORDERKEY BIGINT NOT NULL, L_PARTKEY INTEGER NOT NULL, L_SUPPKEY 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_SHIPPRIORITY INT, L_SHIPMODE VARCHAR(10), L_COMMENT VARCHAR(40), PRIMARY KEY(L_ORDERKEY, L_LINENUMBER));
```

Query executed successfully. Affected rows: 0
Touched 2 sec. Last updated by splice at September 14, 2017, 10:09:37 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:

Import Data

```sql
call SYSCS_UTIL.IMPORT_DATA ('EXAMPLE', 'LINEITEM', null, 's3a://splice-benchmark-data/flat/TPCH/1/lineitem', 'i', null, null, null, null, 0, '/tmp', , true, null);
```

Touched 19 sec. Last updated by anonymous at April 24, 2017, 6:56:51 AM.

Count the number of records in the table

```sql
SELECT COUNT(*) AS NUM_RECORDS FROM EXAMPLE.LINEITEM;
```

Touched 1 sec. Last updated by anonymous at April 24, 2017, 6:57:37 AM.
Collect statistics, to improve query planning, and then run a query:

Create some indexes

```
xsplicemachine
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
);
```

Run compaction

```
xsplicemachine
call SYSYS_UTIL (SYSCS_PERFORM_MAJOR_COMPACTION ON SCHEMA("EXAMPLE");
```

Collect Statistics

```
xsplicemachine
analyze schema EXAMPLE;
```

Run a query

```
xsplicemachine
select
  l_returnflag,
  l_linestatus,
  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_linestatus
order by
  l_returnflag,
  l_linestatus
```

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:
Release Notes for the Splice Machine Database-as-a-Service Product

This topic includes any release notes that are specific to the Splice Machine Database-as-a-Service product, in these sections:

- Current Release Notes
- Major Updates in Previous Releases
- Features Not Yet Available
- Current Limitations
- Important Notes

Current Release Notes
The current AWS and Azure versions of our Splice Machine Database-as-a-Service product are running release 2.7.0.1915 of the Splice Machine database.

These are the major updates in the current version, which was released on August 20, 2019:

<table>
<thead>
<tr>
<th>Category</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Features</td>
<td>Added the ability to make another user a Primary user.</td>
</tr>
<tr>
<td></td>
<td>SSL and Authentication were enabled for the Spark user interface.</td>
</tr>
<tr>
<td></td>
<td>Support was added for multiple %splicemachine interpreter settings.</td>
</tr>
<tr>
<td>Bug Fixes</td>
<td>An issue with the password not being correctly set is fixed.</td>
</tr>
</tbody>
</table>

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.

**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.

**Major Updates in Previous Releases**

The following table summarizes the major updates in the previous release of the Splice Machine Database-as-a-Service product:

<table>
<thead>
<tr>
<th>Category</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Features</td>
<td>Dedicated HDFS is now available.</td>
</tr>
<tr>
<td></td>
<td>Slider added for Notebook Spark.</td>
</tr>
<tr>
<td>Bug Fixes</td>
<td>The display of the memory graph is fixed.</td>
</tr>
<tr>
<td></td>
<td>The display of the CPU graph is fixed.</td>
</tr>
<tr>
<td></td>
<td>An issue with exporting to S3 has been fixed.</td>
</tr>
</tbody>
</table>
Splice Machine Developer’s Guide

This chapter describes how to do development work with your Splice Machine Database. It contains how-to topics and tutorials, and is divided into these sections:

- Accessing Data in the Cloud
- Connecting Splice Machine with Business Intelligence Tools
- Connecting to Splice Machine with JDBC
- Connecting to Splice Machine with ODBC
- Functions and Stored Procedures
- Developer Fundamentals
- Indexing
- On-Premise-Database Authentication
- On-Premise-DB Developer Topics
- Securing Your Database
- Splice*Plus (PL/SQL)
- Tuning and Debugging
- Streaming Data
- Using the Database Console

You’ll find complete documentation for some major features in our Best Practices Guide, including Ingesting Data, On-Premise Maintenance, and Using the Native Spark DataSource.

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 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 DBVisualizer to Splice Machine
Connecting SQuirreL to Splice Machine
Connecting Tableau to Splice Machine

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:

- JDBC Access to Splice Machine with Kerberos
- 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
- ODBC Access to Splice Machine with Kerberos
- Connecting with Python and ODBC
- Connecting with C and ODBC

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

Developer Fundamentals
This section contains topics that contain in-depth information about fundamental aspects of working with Splice Machine:

- Running Transactions
Indexing
This section contains topics related to indexing in your database:

- Indexing Large Tables

On-Premise-Database Authentication
This section shows you how to configure authentication for users of the On-Premise-Database version of Splice Machine, in these topics:

- Using Database Authentication
- Using Native Authentication
- Using LDAP Authentication
- Enabling Kerberos Authentication on Your Cluster

On-Premise-DB Developer Topics
This section contains fundamental developer topics that apply only to the Splice Machine On-Premise Database product:

- Using HCatalog
- Using HAProxy with Splice Machine
- Using MapReduce
- Working with HBase
- Using Zeppelin with our On-Premise Database

Securing Your Database
This section shows you how to configure security for use with Splice Machine, in these topics:
Securing Connections with SSL/TLS
Accessing Splice Machine from Windows on a Kerberized Cluster
Authorizing Users and Roles
Summary of Permissions for Users and Roles
Using Apache Ranger with Splice Machine

Splice*Plus (PL/SQL)

Introduction
Using Splice*Plus
The Splice*Plus Language

Tuning and Debugging
This section contains information about tuning the performance of your database, as well as debugging slowdowns, in these topics:

Optimizing Queries
Using Statistics
Using Explain Plan
Explain Plan Examples
Logging
Debugging
Using Snapshots

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

Configuring a Kafka Feed
Creating a Kafka Producer

Using the Database Console
Our Database Console tutorial walks you through using the Splice Machine database console to monitor queries in real time:
» Introduction
» Features of the Database Console
» Managing Queries
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.

This topic contains these two sections:

- Configuring S3 Bucket Access
- Accessing S3 Buckets

### Configuring 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 (Identify and Access Management) service:**
   Select IAM in the **Security, Identity & Compliance** section:
Configuring an S3 Bucket for Splice Machine Access
4. **Create a new policy:**

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

   ![Create Policy Screen](image)

   b. Select **Create Your Own Policy** to enter 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*):

```
Policy Name
splice_access
```

**Description**

Allowing Splice Machine access to our S3 bucket.

```
Policy Document

```

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.
e. Click **Validate Policy** to verify that your policy settings are valid.
f. Click **Create Policy** to create and save the policy.

5. **Add Splice Machine as a user:**
   After you create the policy:
   a. Select **Users** from the left-hand navigation pane.
   b. Click **Add User**.
   c. Enter a **User name** (we've used *SpliceMachine*) and select **Programmatic access** as the access type:
d. Click **Attach existing policies directly**.

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

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.
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.

Accessing S3 Buckets

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

```sql
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:

```sql
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.access.key</name>
  <value>access key</value>
</property>

<property>
  <name>fs.s3a.secret.key</name>
  <value>secret key</value>
</property>
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](https://aws.amazon.com).

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
   a. 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.
Using Azure WASB, ADLS, and ADLS2 with Splice Machine

This topic walks you through the steps required to configure Azure Blob (WASB), Azure Data Lake Storage (ADLS), and Azure Data Lake Storage Gen 2 (ADLS2) with Splice Machine.

This topic contains these sections:

- Configuring WASB and ADLS Storage Access
- Configuring ADLS2 Storage Access

Configuring WASB and ADLS Storage Access

This section shows you how to configure WASB and ADLS storage for Splice Machine Access, in these subsections:

- Configuring and Uploading Your Data
- Copying Data Between WASB and ADLS
- Importing Your Data from Azure WASB or ADLS Storage

Configuring and Uploading Your Data

1. Log in to the Azure portal.
   If needed, first create an Azure account. Then log in to the portal at https://portal.azure.com.

2. Create a resource group:
   Follow the instructions on this page to create a resource group: https://docs.microsoft.com/en-us/azure/azure-resource-manager/manage-resource-groups-portal.
   For example, create the myResourceGroup resource group.

3. Create a Data Lake Storage Gen1 account:
   For example:

   ```
   name: myadls1
   resource group: myResourceGroup
   ```

4. Upload your data file:
   In the Azure Data Explorer, create a new folder, and then upload your file (for example, myData) to that folder.
5. Create credentials:

   - Register an AD application: myApp
   - Assign the application to a role: Owner
   - Copy IDs:
     - Tenant ID: <tenantID>
     - Client ID: <clientID>
   - Create a new application secret
     - myApp secret: <clientSecret>

6. Assign the Azure AD application to the Azure Data Lake Storage Gen1 account file.

7. Get the OAuth 2.0 token endpoint:
   For example: [https://login.microsoftonline.com/<tenantID>/oauth2/token](https://login.microsoftonline.com/<tenantID>/oauth2/token)

8. Access ADLS from your cluster:
   To access your ADLS storage from Cloudera, follow the instructions in this page: [https://www.cloudera.com/documentation/enterprise/5-12-x/topics/admin_adls_config.html](https://www.cloudera.com/documentation/enterprise/5-12-x/topics/admin_adls_config.html). For example:

   ```
   hadoop fs -Ddfs.adls.oauth2.access.token.provider.type=ClientCredential \
   -Ddfs.adls.oauth2.client.id=<clientID> \
   -Ddfs.adls.oauth2.credential="<clientSecret>" \
   -Ddfs.adls.oauth2.refresh.url=https://login.microsoftonline.com/<tenantID>/oauth2/token \
   -ls adl://myadls1.azuredatalakestore.net/myData
   ```

9. Create a WASB storage account:
   Follow the instructions on this page: [https://docs.microsoft.com/en-us/azure/storage/common/storage-quickstart-create-account](https://docs.microsoft.com/en-us/azure/storage/common/storage-quickstart-create-account).

   - Resource group: myResourceGroup
   - Storage account name: mywasb
   - Get access key: <accessKey>
   - Create container: myData

**Copying Data Between WASB and ADLS**

Importing Your Data from Azure WASB or ADLS Storage

If you’re using Cloudera, to import your data into Splice Machine, you need to add property values to this file:

```
HDFS->Configuration->Cluster-wide Advanced Configuration Snippet (Safety Valve) for core-site.xml
```

Add the following values:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dfs.adls.oauth2.access.token.provider.type</td>
<td>ClientCredential</td>
</tr>
<tr>
<td>dfs.adls.oauth2.client.id</td>
<td>&lt;clientID&gt;</td>
</tr>
<tr>
<td>dfs.adls.oauth2.credential</td>
<td>&lt;clientSecret&gt;</td>
</tr>
<tr>
<td>dfs.adls.oauth2.refresh.url</td>
<td><a href="https://login.microsoftonline.com/">https://login.microsoftonline.com/</a>&lt;tenantID&gt;/oauth2/token</td>
</tr>
<tr>
<td>fs.azure.account.key.mywasb.blob.core.windows.net</td>
<td>&lt;accessKey&gt;</td>
</tr>
</tbody>
</table>

Then, you can import data with statements like the following:

```
splice> call SYSCS_UTIL.IMPORT_DATA('mySchema', 'myTable1', null, 'wasbs://myData@mywasb.blob.core.windows.net/myTbl.tbl', '|', null, null, null, null, 0, '/BAD', true, null);
splice> call SYSCS_UTIL.IMPORT_DATA('mySchema', 'myTable2', null, 'adl://myadls1.azuredatalakestore.net/myData/myTbl.tbl', '|', null, null, null, null, 0, '/BAD', true, null);
```
Configuring ADLS2 Storage Access


You can currently use ADLS2 with:

- Hadoop 3.2+
- Cloudera 6.1+
- HortonWorks 3.1.x+

The remainder of this section you how to configure ADLS2 storage for Splice Machine Access, in these subsections:

- Configuring ADLS2 for Splice Machine
- Copying Your Data from WASB to ADLS2
- Importing Your Data from Azure ADLS2

## Configuring ADLS2 for Splice Machine

1. Log in to the Azure portal:
   If needed, first create an Azure account. Then log in to the portal at [https://portal.azure.com](https://portal.azure.com).

2. Create a resource group:
   
   For example, create the `myResourceGroup` resource group.

3. Create an ADLS2 storage account:
   
   For example:

   ```
   Name: myadls2
   Resource group: myResourceGroup
   Location: West US 2
   Account kind: StorageV2
   ```

4. Manage the storage account:
   - Add a file system; for example `myData`.
   - Get access key: `<accessKeyADLS2>`
Copying Your Data from WASB to ADLS2
For Cloudera, follow the instructions in this page: https://www.cloudera.com/documentation/enterprise/latest/topics/admin_adls2_config.html. For example:

```
hadoop fs \
-Dfs.azure.account.key.olegadls2.dfs.core.windows.net="<accessKeyADLS2>" \
-Dfs.azure.account.key.olegwasb.blob.core.windows.net="<accessKeyWASB>" \
-cp wasbs://tpch1@olegwasb.blob.core.windows.net/* abfs://tpch1@olegadls2.dfs.core.windows.net/
```

Importing Your Data from Azure ADLS2
If you're using Cloudera, to import your data into Splice Machine, you add property values to this file:

HDFS->Configuration->Cluster-wide Advanced Configuration Snippet (Safety Valve) for core-site.xml

Add this value:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fs.azure.account.key.myadls2.dfs.core.windows.net</td>
<td>&lt;accessKeyADLS2&gt;</td>
</tr>
</tbody>
</table>

You can then import data with a statement like the following:

```
splice> call SYSCS_UTIL.IMPORT_DATA('SPLICE', 'CUSTOMER', null, 'abfs://myData@myadls2.dfs.core.windows.net/myTable.tbl', '|', null, null, null, null, 0, '/BAD', true, null);
```
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 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 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.

You can read more about our JDBC Driver here. And you can download the driver from here: https://www.splicemachine.com/get-started/jdbc-driver-download/.

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:
c. Enter the following URL into the Database URL field that appears once you've selected the driver:

```
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
```

d. Fill in the Userid and Password fields with your user Id and password. 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 website (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.

You can read more about our JDBC Driver here. And you can download the driver from here: https://www.splicemachine.com/get-started/jdbc-driver-download/.

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:

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.

![Open Window](image)

Name          | Date Modified
--------------|---------------
`db-client-2.5.0.1708-SNAPSHOT.jar` | Monday, February 27, 2017 4:00 PM

- d. Now, back in the Add Driver screen, click the List Drivers button to 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

- a. Click theAliases tab in the SQuirreL window, and then click the Create new Alias (blue + sign) button.

- b. 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 your user ID as the **User:** value and your password for the **Password:** value.
d. Click the Connect button to verify your connection. You should see the success message:
Connection successful
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](http://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.

You can read more about our JDBC Driver here. And you can download the driver from here: [https://www.splicemachine.com/get-started/jdbc-driver-download/](https://www.splicemachine.com/get-started/jdbc-driver-download/).

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 (SPICE) as the connection name for use in Tableau.

4. After you click OK, Tableau is ready to work with 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; typically:

/splicemachine/jdbc-driver/splicemachine/jdbc-driver/

You can also download the driver from our JDBC Download site:


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.

<table>
<thead>
<tr>
<th>Connection Parameter</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>1527</td>
</tr>
<tr>
<td>User name</td>
<td>Substitute your own user ID</td>
</tr>
<tr>
<td>Connection Parameter</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Password</td>
<td>Substitute your own password</td>
</tr>
<tr>
<td>Database name</td>
<td>splicedb</td>
</tr>
</tbody>
</table>
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:
   
   ```bash
   /splicemachine/jdbc-driver/splicemachine/jdbc-driver/
   ```

   You can also download the driver from our JDBC Download site:

   ```url```
   https://www.splicemachine.com/get-started/jdbc-driver-download/

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[] arg) {
        //JDBC Connection String - sample connects to local database
        String dbUrl = "jdbc:splice://localhost:1527/splicedb;user=YourUserId;password=YourPassword";
        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:

<table>
<thead>
<tr>
<th>Connection Parameter</th>
<th>Default Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC URL</td>
<td>jdbc:splice://&lt;hostname&gt;:1527/splicedb</td>
<td>Use localhost as the &lt;hostname&gt; value for the standalone version of Splice Machine. On a cluster, specify the IP address of an HBase RegionServer.</td>
</tr>
<tr>
<td>JDBC driver class</td>
<td>com.splicemachine.db.jdbc.ClientDriver</td>
<td></td>
</tr>
</tbody>
</table>

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:

```
```

The command should display a result like the following:

```
record=[1] a=[1] b=[a]
```
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

You can read more about our JDBC Driver here. And you can download the driver from here: https://www.splicemachine.com/get-started/jdbc-driver-download/.

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 and URL values:

   ```
   # Configure Using Gemfile
   # gem 'activerecord-jdbcsqlite3-adapter'
   # gem 'activerecord-jdbcderby-adapter'
   #
   development:
   adapter: jdbcderby
database: splicedb
username: yourUserId
password: yourPassword
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`. 

Splice Machine Documentation

Connecting to Splice Machine with JRuby and JDBC
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:

```sql
splice> describe SPLICE.MYTESTTABLE;
COLUMN_NAME | TYPE_NAME | DEC | NUM | COLUMN | COLUMN_DEF | CHAR_OCTE | IS_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=YourUserId;password=YourPassword");
  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:

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

You can read more about our JDBC Driver here. And you can download the driver from here: https://www.splicemachine.com/get-started/jdbc-driver-download/.

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=/splicemachine/jdbc-driver/splicemachine/jdbc-driver/
   ```

2. Copy the example program code:
   You can copy and paste the code below; note that this example uses our default connectivity database and URL parameters:
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', 'yourUserId')
props.setProperty('password', 'yourPassword')
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=[9] id=[c1e0c303-0144-fcd8-d703-00003cbfba48] name=[TESTOUTPUT2] 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:

You can read more about our JDBC Driver here. And you can download the driver from here: https://www.splicemachine.com/get-started/jdbc-driver-download/.

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
   >>> 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.

You can read more about our JDBC Driver here. And you can download the driver from here: https://www.splicemachine.com/get-started/jdbc-driver-download/.

NOTE: The steps in this topic 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", "yourUserId", "yourPassword")
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

You can read more about our JDBC Driver here. And you can download the driver from here: https://www.splicemachine.com/get-started/jdbc-driver-download/.

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=/splicemachine/jdbc-driver/splicemachine/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=YourUserId;password=YourPassword"

        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 in t, 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()

        } finally {

            // close the connection
            if (connection != null) {
                connection.close()
            }
        }
    }
}


3. Save the code to SampleScalaJDBC.scala.

4. Compile the program:

```bash
scalac SampleScalaJDBC.scala
```
5. **Run the program:**

Run the `SampleScalaJDBC` program as follows:

```
scala SampleScalaJDBC
```

The command should display a result like the following:

```
```
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.

You can read more about our JDBC Driver here. And you can download the driver from here: https://www.splicemachine.com/get-started/jdbc-driver-download/.
JDBC Access to Splice Machine with Kerberos

This section shows you how to connect your applications to Splice Machine on a Kerberized cluster, using our JDBC driver. As a prerequisite to connecting, you must ensure that:

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

See Enabling Kerberos Authentication for information about using Splice Machine on a Kerberized cluster.

You can read more about our JDBC Driver here. And you can download the driver from here: https://www.splicemachine.com/get-started/jdbc-driver-download/.

Connecting Splice Machine with Kerberos and JDBC

Once you've configured Kerberos, you can connect with JDBC by specifying the principal and keytab values in your connection string; for example:

splice> CONNECT 'jdbc:splice://localhost:1527/splicedb;principal=jdoe@SPLICEMACHINE.Colo;keytab=/tmp/user1.keytab';

If you're using HAProxy, simply specify your proxy host as the server in the connect string:

splice> CONNECT 'jdbc:splice://<haproxy-host>:1527/splicedb;principal=jdoe@SPLICEMACHINE.Colo;keytab=/tmp/user1.keytab';

If your keytab file is stored on HDFS, you can specify the connection like this instead:

splice> CONNECT 'jdbc:splice://localhost:1527/splicedb;principal=jdoe@SPLICEMACHINE.Colo;keytab=hdfs:///tmp/splice.keytab';

When connecting third-party software via JDBC using a keytab file stored on HDFS, you must make sure that the Splice Machine libraries are in your classpath:

export HADOOP_CLASSPATH=/opt/cloudera/parcels/SPLICEMACHINE/lib/*
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
- Configuring Kerberos for Windows
- 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:

<table>
<thead>
<tr>
<th>Connection Parameter</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>1527</td>
</tr>
<tr>
<td>User name</td>
<td>Substitute your own user ID.</td>
</tr>
<tr>
<td>Password</td>
<td>Substitute your own password.</td>
</tr>
<tr>
<td>Database name</td>
<td>splicedb</td>
</tr>
</tbody>
</table>
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 on 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:
You can use either Microsoft Active Directory Kerberos or MIT Kerberos for your ODBC connections; see our Accessing Splice Machine on a Kerberized Cluster topic for instructions.

**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 Advanced Options**
Click the Advanced options button to set advanced properties:
Use the *Driver Properties* tab to configure logging options that apply to all connections using this driver:

Logging driver activity can be handy for debugging connection issues; however, it adds overhead and will have a significant impact on performance.
Use the *Connection Properties* tab to configure SSL and network options.

### Including the Catalog Name
The *Include catalog name in table metadata* option is available to address an issue with certain applications that expect a catalog entry in the table metadata. When this option is selected, our driver returns the catalog name in the table metadata but sets the `catalog usage` attribute to 0 to tell applications that they shouldn't use the catalog name in a Splice Machine database query. The Splice Machine database considers it an error when a query uses a fully qualified table name containing the catalog (e.g. `splicedb.SPLICE.TABLENAME`), so you can use this driver feature to also strip any `splicedb` catalog names out of the query.

If you see an application that warns of illegal or missing catalog names, or shows an empty catalog, try selecting this option.

### 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:
<table>
<thead>
<tr>
<th>SSL Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>The communications channel is encrypted, but no attempt is made to verify client or host certificates.</td>
</tr>
<tr>
<td>peerAuthentication</td>
<td>You must specify the location of both the client certificate file and the client private key in their respective fields.</td>
</tr>
<tr>
<td></td>
<td>The Certificate file field defaults to the .pem 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.</td>
</tr>
<tr>
<td></td>
<td>The Private key file field defaults to the .key 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.</td>
</tr>
</tbody>
</table>

You can find more information about PEM files by searching the web for pem formatted file.

Select the Always trust server certificate 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.

**NOTE:** You should select the Always trust server certificate option if you are using a self-signed certificate.

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.

## Installing the Driver on Linux

Follow these steps to install the Splice Machine ODBC driver on a Linux computer:

1. **Make sure you have the software our driver requires installed:**
   - You must have version 4.82 or later of the GNU Compiler Collection (GCC) installed. You can use the following command to verify your version:
     ```
     gcc --version
     ```
   - You must have version 2.2.12 or later of the unixODBC driver manager installed. You can use the following command to verify your version:
     ```
     odbcinst -j
     ```
   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](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/](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_linux64-<version>.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_linux64-<version>.tar.gz
   ```
   This creates a directory named splice_odbc_64.

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:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib64</td>
<td>The driver binary.</td>
</tr>
<tr>
<td>errormessages</td>
<td>The XML error message source for any error messages issued by the driver.</td>
</tr>
</tbody>
</table>

4. **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.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>odbc.ini</td>
<td>Specifies the ODBC data sources (DSNs).</td>
</tr>
<tr>
<td>odbcinst.ini</td>
<td>Specifies the ODBC drivers.</td>
</tr>
<tr>
<td>splice.odbcdriver.ini</td>
<td>Configuration information specific to the Splice Machine ODBC driver.</td>
</tr>
</tbody>
</table>

The default version of the odbc.ini file looks like this:
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.

5. **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
You can specify one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OFF</td>
</tr>
<tr>
<td>1</td>
<td>LOG_FATAL</td>
</tr>
<tr>
<td>2</td>
<td>LOG_ERROR</td>
</tr>
<tr>
<td>3</td>
<td>LOG_WARNING</td>
</tr>
<tr>
<td>4</td>
<td>LOG_INFO</td>
</tr>
<tr>
<td>5</td>
<td>LOG_DEBUG</td>
</tr>
<tr>
<td>6</td>
<td>LOG_TRACE</td>
</tr>
</tbody>
</table>

The larger the LogLevel value, the more verbose the logging.

Logging does impact driver performance.

### LogPath
The path to the directory in which you want the logging files stored. Two log files are written in this directory:

- **splice_driver.log**
  - Contains driver interactions with the application and the driver manager.

- **splice_derby.log**
  - Contains information about the drivers interaction with the Splice Machine cluster.

After configuring logging, copy the file to `/etc`:

### 6. Verify your installation
You can test your installation by using the following command to run `isql`:

```
isql SpliceODBC64 yourUserId yourPassword
```
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.iodbc.org/](http://www.iodbc.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/](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_macosx64-2.5-51.0.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_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:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>odbc.ini</td>
<td>Specifies the ODBC data sources (DSNs).</td>
</tr>
<tr>
<td>odbcinst.ini</td>
<td>Specifies the ODBC drivers.</td>
</tr>
<tr>
<td>splice.odbcdriver.ini</td>
<td>Configuration information specific to the Splice Machine ODBC driver.</td>
</tr>
</tbody>
</table>

   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_odbc.so
UID          = yourUserId
PWD          = yourPassword
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
ErrorMessagePath=/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:
To configure logging, modify the `LogLevel` and `LogPath` values:

<table>
<thead>
<tr>
<th>LogLevel</th>
<th>You can specify one of the following values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OFF</td>
</tr>
<tr>
<td>1</td>
<td>LOG_FATAL</td>
</tr>
<tr>
<td>2</td>
<td>LOG_ERROR</td>
</tr>
<tr>
<td>3</td>
<td>LOG_WARNING</td>
</tr>
<tr>
<td>4</td>
<td>LOG_INFO</td>
</tr>
<tr>
<td>5</td>
<td>LOG_DEBUG</td>
</tr>
<tr>
<td>6</td>
<td>LOG_TRACE</td>
</tr>
</tbody>
</table>

The larger the `LogLevel` value, the more verbose the logging.

Logging does impact driver performance.

<table>
<thead>
<tr>
<th>LogPath</th>
<th>The path to the directory in which you want the logging files stored. Two log files are written in this directory:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>splice_driver.log</code> contains driver interactions with the application and the driver manager</td>
</tr>
<tr>
<td></td>
<td><code>splice_derby.log</code> contains information about the drivers interaction with the Splice Machine cluster</td>
</tr>
</tbody>
</table>

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:
   ```c
   #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**
   ```bash
   #!/bin/bash
   # gcc -I /usr/local/splice/unixODBC/include listODBCdriver.c -o listODBCdriver
   # gcc -I /usr/local/splice/unixODBC/include -L /usr/local/splice/lib -lodbc -lodbcinst -lodbcsrc
   ```

3. **Run the program**
   Run the compiled listODBCdriver:
   ```bash
   prompt:$ ./listODBCdriver
   
   The command should display a result like the following:
   Splice Machine - Description=Splice Machine ODBC Driver
ODBC Access to Splice Machine with Kerberos

This section shows you how to connect your applications to Splice Machine on a Kerberized cluster, using our ODBC driver. As a prerequisite to connecting, you must ensure that:

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

See Enabling Kerberos Authentication for information about using Splice Machine on a Kerberized cluster.

Connecting Splice Machine with Kerberos and ODBC

Follow these steps to connect to a Kerberized cluster with ODBC:

1. Follow our instructions for installing and configuring our ODBC driver. Verify that the odbc.ini configuration file for the DSN you’re connecting to includes this setting:

   USE_KERBEROS=1

2. Establish a default security principal user with a ticket-granting ticket (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 connects using ODBC; our ODBC driver will use that default Kerberos *principal* when authenticating with Splice Machine.
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 and 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:
   
   ```shell
   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):

   ```python
   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
table_name = cursor.fetchone()

# If there is a record, print it
if table_name:
    print(table_name)

# The following will continue to retrieve one record at a time from the resultset
while 1:
    table_name = cursor.fetchone()
    if not table_name:
        break
    print(f'table name: {table_name.TABLENAME}')

# The following is an example of using the fetchall option, instead of retrieving one record at time
cursor.execute("select * from SYS.SYSSCHEMAS")
rows = cursor.fetchall()
for row in rows:
    print(f"row.SCHEMADID: {row.SCHEMADID}, row.SCHEMANAME: {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:

   ```c
   #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**


```bash
#!/bin/bash
# gcc -I /usr/local/splice/unixODBC/include listODBCdriver.c -o listODBCdriver -L/usr/local/splice/lib -lodbc -lodbcinst -lodbcrr
```

### 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
```
Using Attunity to Export MySQL Tables to Splice Machine

Follow these steps to use *Attunity Replicate* to export tables from a MySQL database and import them into your Splice Machine database:

1. Create a Splice Machine Endpoint in Attunity
2. Create an Attunity Task to Export Data
3. Run the Replication Task
4. Import the Data into Splice Machine

1. Create a Splice Machine Endpoint in Attunity

Your first step is to open the *Attunity Replicate* user interface and click **Manage Endpoint Connections**.

In the pop-up window that opens, select **+ New Endpoint Connection**, and follow these steps:

1. Enter data in these fields:
   a. Enter the name for your endpoint in the **Name** field.
   b. Enter a description for the endpoint in the **Description** field.
   c. Select **Target** as the value of the **Role** field.
   d. Enter **Hadoop** as the value of the **Type** field. You'll now see three new fields displayed: **Security**, **HDFS**, and **Hive Access**.

2. Click the down-arrow to fill in **Security** information:
   a. Select **User name** in the **Authentication type** field.
   b. Enter the Hadoop username in the **Username** field. For example: *hdfs*.

3. Click the down-arrow to fill in **HDFS** information:
   a. Select **WebHDFS** in the **Use** field.
   b. Enter the IP or host name of a Hadoop name node in the **NameNode** field.
   
   Make sure all of the data nodes are accessible from Attunity and configure all of the Hadoop nodes in `/etc/hosts` file in the machine that is running Attunity.
   
   c. Enter the port to use for webHDFS in the **Port** field. The default port value is 50070.
   
   Make sure the WebHDFS service is running.
   
   d. Enter your preferred target location in the **Target folder** field. Make sure that the user you
specified in the **Username:** field has write access to this folder.<p>

4. Click the down-arrow to fill in the **Hive Access** information:
   
   a. Select **No Access** in the **Access Hive using:** field.

5. Now select the **Advanced** tab near the top of the window, and click the down-arrow to display the **File Format** settings:
   
   a. Change the **Field Delimiter:** character to |.
   
   b. Click the **Save** button at the bottom of the window to save your endpoint settings.
   
   c. Click the **Test Connection** button to test your settings.

---

### 2. Create an Attunity Task to Export Data

Now that you've created the endpoint, you need to create an Attunity Replicate task that exports data from the MySQL source to files that Splice Machine can import.

Start by clicking **+ New Task** in the Attunity Replicate UI, and then follow these steps:

1. Enter the name for your task in the **Name:** field.

2. Enter a description for the task in the **Description:** field.

3. Select **Unidirectional** in the **Replication Profile** field.

4. In the **Task Options** section:
   
   ▷ Select **Full Load**.
   
   ▷ Select **Apply Changes**.
   
   ▷ If you want to store the change logs on the target you selected, select **Store Changes**.

5. Click **OK** to save the task settings and proceed to the **Table Selection** settings.

6. Drag and drop the source and target endpoints into the center panel. Select the **Splice Machine** endpoint you just created as the target endpoint.
   
   a. Click **Table Selection** to open the pop-up window for selecting tables.
   
   b. Select the tables from your source endpoint
   
   c. Click **OK**.

7. Click **Save** in the top-left portion of the window to save the task.
3. Run the Replication Task
Now you can run your newly defined replication task by clicking Run.

Once your task is running, you can:

» Click Monitor to monitor the task.
» Click Stop to stop the task.

4. Import the Data into Splice Machine
Once the files have been transferred use one of the ingest methods available in Splice Machine to import the data into your database.
Using Attunity via ODBC to Export to Splice Machine

Follow these steps to use Attunity Replicate with the Splice Machine ODBC driver to export tables from a MySQL database and import them into your Splice Machine database:

1. **Install the Splice Machine ODBC Driver**
2. **Create a Splice Machine Endpoint in Attunity**
3. **Create an Attunity Task to Export Data**
4. **Run the Replication Task**
5. **Verify the Tables in Splice Machine**

### 1. Install the Splice Machine ODBC Driver

If you don't already have our ODBC driver installed, please follow the instructions in our Installing the Splice Machine ODBC Driver on Linux topic to install and test the driver on your system.

### 2. Create a Splice Machine Endpoint in Attunity

Your first step is to open the Attunity Replicate user interface and click Manage Endpoint Connections.

In the pop-up window that opens, select + New Endpoint Connection, and follow these steps:

1. Enter data in these fields:
   a. Enter the name for your endpoint in the **Name:** field.
   b. Enter a description for the endpoint in the **Description:** field.
   c. Select **Target** as the value of the **Role:** field.
   d. Select **ODBC** as the value of the **Type:** field.

2. Click the **Connection String** button, and enter ODBC connection information. Specify the Driver exactly as shown here, and replace the \`Port` \`and \`URL` values as required for your environment:

   ```sql
   Driver={SpliceODBCDriver};Port=1527;URL=192.168.2.215
   ```

3. Enter your Splice Machine user name and password in the **Username** and **Username** fields.
4. Click the **Save** button at the bottom of the window to save your endpoint settings.
5. Click the **Test Connection** button to test your settings.
3. Create an Attunity Task to Export Data
Now that you've created the endpoint, you need to create an Attunity Replicate task that exports data from the MySQL source to files that Splice Machine can import.

Start by clicking **+ New Task** in the Attunity Replicate UI, and then follow these steps:

1. Enter the name for your task in the **Name:** field.
2. Enter a description for the task in the **Description:** field.
3. Select **Unidirectional** in the **Replication Profile** field.
4. In the **Task Options** section:
   - Select **Full Load**.
   - Select **Apply Changes**.
   - If you want to store the change logs on the target you selected, select **Store Changes**.
5. Click **OK** to save the task settings and proceed to the **Table Selection** settings.
6. Drag and drop the source and target endpoints into the center panel. Select the **Splice Machine** endpoint you just created as the target endpoint.
   a. Click **Table Selection** to open the pop-up window for selecting tables.
   b. Select the tables from your source endpoint
   c. Click **OK**.
7. Click **Save** in the top-left portion of the window to save the task.

4. Run the Replication Task
Now you can run your newly defined replication task by clicking **Run**.

Once your task is running, you can:

- Click **Monitor** to monitor the task.
- Click **Stop** to stop the task.

5. Verify the Tables in Splice Machine
Once the replication task has completed, you can use the **splice>** command line interpreter (**sqlshell.sh**) to verify that the tables look good in your Splice Machine database:
To verify that the new schema you replicated from Attunity, display a list of available schemas:

splice> SHOW SCHEMAS;

To see a list of the tables in your schema, use the SHOW TABLES command. For example, if your schema name is mySchema, use this command:

splice> SHOW TABLES IN MYSHEMA;

If you selected the Store Changes options in your replication task settings, you’ll see change tables listed with a format like this:

<MY_TABLE>__ct

To verify the number of rows in a table, use a command like this:

splice> SELECT COUNT(*) FROM MYSHEMA.MYTABLE;

To display the first 10 rows of a table, use a command like this:

splice> SELECT * FROM MYSHEMA.MYTABLE {LIMIT 10};

Though splice> command lines are shown in all caps in the examples above, the commands are not case sensitive.
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 (UDFs) 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` statements. 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:

```sql
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

A Splice Machine database function:

- 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

Stored Procedure

Splice Machine stored procedures can:

- 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

Can be used in SELECT statements.

Cannot be used in SELECT statements.

Must be invoked using a CALL statement.

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.

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.

---

**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:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Functions</th>
<th>Stored Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute in an SQL Statement</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Execute in a Trigger</td>
<td>Yes</td>
<td>Triggers that execute before an operation (before triggers) cannot modify SQL data.</td>
</tr>
<tr>
<td>Process OUT / INOUT Parameters</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Return ResultSet(s)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Operation</td>
<td>Functions</td>
<td>Stored Procedures</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------</td>
<td>-------------------</td>
</tr>
<tr>
<td>Execute SQL Select</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Execute SQL Update/Insert/Delete</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Execute DDL (Create/Drop)</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>splice&gt; show functions;</code></td>
<td>All functions defined in your database</td>
</tr>
<tr>
<td><code>splice&gt; show functions in SYSCS_UTIL;</code></td>
<td>All functions in the SYSCS_UTIL schema in your database</td>
</tr>
<tr>
<td><code>splice&gt; show procedures;</code></td>
<td>All stored procedures defined in your database</td>
</tr>
<tr>
<td><code>splice&gt; show procedures in SYSCS_UTIL;</code></td>
<td>All stored procedures in the SYSCS_UTIL schema in your database</td>
</tr>
</tbody>
</table>

**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

This section includes these subsections:

- Writing a Function in JAVA
- Writing a Function in Python

Writing a Function in Java

Follow the steps below to write a Splice Machine database function in Java.

1. **Create a Java method**
   Each function maps to a Java method. For example:

   ```java
   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 addNum(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 Function in Python**

Follow the steps below to write a Splice Machine database function in Python.

> Creating functions in Python is currently a **Beta Release** feature; it will become generally available in a future release.

**1. Define a Python script as a function in your database:**

```python
CREATE FUNCTION SPLICE.PYSAMPLE_FUNC( a VARCHAR(50) )  
    RETURNS VARCHAR(50)  
    PARAMETER STYLE JAVA  
    READS SQL DATA  
    LANGUAGE PYTHON  
    AS 'def run(inputStr):  
        import re  
        result = inputStr.strip().split(',')[0]  
        return result';
```

You can find the complete syntax for CREATE FUNCTION in the *Splice Machine SQL Reference* manual.
2. **Invoke your function**

You can invoke functions just like you would call any built-in database function. For example, you could use the above sample function as follows:

```
splice> VALUES SPLICE.PYSAMPLE_FUNC('Splice,Machine');
1
```

Writing a Stored Procedure in Splice Machine

This section includes these subsections:

- [Writing a Stored Procedure in JAVA](#)
- [Writing a Stored Procedure in Python](#)

**Writing a Stored Procedure in JAVA**

Follow the steps below to write a stored procedure in Java.

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 the database
     */
    public static void GET_TABLE_NAMES(ResultSet[] rs) throws SQLException {
        Connection conn = DriverManager.getConnection("jdbc:default:connection");
        PreparedStatement pstmt = conn.prepareStatement("select * from sys.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-SNAPSHOT.jar',
     'SPLICE.CUSTOM_SPLICE_PROCS_JAR', 0);
   CALL SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY('derby.database.classpath', 'SPLICE.CUSTOM_SPLICE_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.
Writing a Stored Procedure in Python

Follow the steps below to write a stored procedure in Python. Please note the following Python-related version information:

- Creating functions in Python is currently a Beta Release feature; it will become generally available in a future release.
- Our Python stored procedure implementation uses Jython 2.5.3.
- Python scripts should be compatible with Python version 2.5.
- The JDBC connection uses Python DB API 2.0; it is implemented by zxJDBC.

Specifying Your Script

When creating a Python stored procedure, include your Python script directly in the `CREATE PROCEDURE` statement. Here's a simple example:

```
splice> CREATE PROCEDURE SPLICE.PYTHON_TEST (  
    IN limit INT )  
PARAMETER STYLE JAVA  
LANGUAGE PYTHON  
DYNAMIC RESULT SETS 1  
READS SQL DATA  
AS 'def run(lim, res):  
    c = conn.cursor()  
    # select alias and javaclassname columns from sys.sysaliases tables  
    # return them as a ResultSet  
    stmt = "select alias, javaclassname from sys.sysaliases {limit ?}"  
    c.executemany(stmt,[lim])  
    d = c.description  
    result = c.fetchall()  
    # construct the ResultSet and fill it into the ResultSet list  
    res  
    res[0] = factory.create([d,result])  
    conn.commit()  
    c.close()  
    conn.close()';  
0 rows inserted/updated/deleted
```
General Rules
Here are some important notes about your script:

- The entire script must be enclosed in single quotes.
- Use double quotes (") around strings within the script; if you must use single quote within the script, specify each as two single quotes ('').
- Use spaces instead of tabs within your scripts; the command line processor will convert tabs to a single space in your script, even within a string.
- Write the script under the run function.
- The arguments you specify for your script in the CREATE PROCEDURE statement should match the order specified in your method definition. Note that their names do not need to match.

Connecting to Your Database
The conn global variable provides a default connection to your database.

Restrictions
Transactional auto-commit cannot be enabled within SQL statements in your stored procedure; this is due to the fact that the SQL statements are executed via a nested connection.

Constructing and Returning ResultSets
If your procedure returns a ResultSet, you must specify the ResultSet as a final argument to your method; for example, res in this snippet:

```python
def run(lim, res)
...
d = c.description
result = c.fetchall()
res[0] = factory.create([d,result])
```

You can use the pre-defined create function to construct the ResultSet. Access this function from the global variable factory, which is defined here:

```python
com.splicemachine.derby.impl.sql.pyprocedure.PyStoredProcedureResultSetFactory
```

The factory.create function has the following syntax:

```python
factory.create( description, resultRows)
```

*description*
A tuple containing these 7 values:
Running and Updating Your Procedure

See Running Your Stored Procedure for information about running your stored procedure, and Updating Your Stored Procedure for information about updating your stored procedure.

Running Your Stored Procedure

You can run your stored procedure by calling it from the splice> prompt. For example:

```
splice> call SPLICE.GET_TABLE_NAMES();
splice> call SPLICE.PYTHON_TEST(5);
```

Updating 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-SNAPSHOT.jar',
  'SPLICE.CUSTOM_SPLICE_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 ExampleStoredProcedure class:

```java
public class ExampleStoredProcedure {
    public static void MY_TEST_PROC(String myInput, ResultSet[] resultSet) throws SQLException {
        ...
    }
}
```

Here are a set of things you should know about ResultSet[] in stored procedures:

- Although the CREATE PROCEDURE call allows you to specify the number of DYNAMIC RESULT SETS, we currently only support returning a single ResultSet.

- 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.
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

```sql
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:
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:

```sql
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**

```sql
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**

```sql
splice> CALL SQLJ.REMOVE_JAR('SPLICE.MY_EXAMPLE_APP', 0);
```
Please refer to the \texttt{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 \texttt{schema.name} format.
- The second argument is currently unused but required; use 0 as its value.

\textbf{NOTE:} The Jar file operations (the \texttt{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 (a UDF, or user-defined 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`:

   ```java
   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 the beach.", 5
     * will return "Today is a wonderful day".
     * @param inboundSentence
     * @param numberOfWords
     * @return
     */
    public static String wordLimiter(String inboundSentence, int numberOfWords) {
        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:
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:
2. **Create the shell of our Java class.**
We create a class named `ExampleStringFunctions` in the package `com.splicemachine.examples`.

```java
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.

```java
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[] resultSet) throws SQLException {
        try {
            Connection con = DriverManager.getConnection("jdbc:default:connection");
            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');
```
Splice Machine Fundamentals

This section contains the following fundamental topics for Splice Machine developers:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Describes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Running Transactions</strong></td>
<td>Introduces you to the basics of running transactions with Splice Machine.</td>
</tr>
<tr>
<td><strong>Working with Dates</strong></td>
<td>Provides an overview of working with date and time values in Splice Machine.</td>
</tr>
<tr>
<td><strong>Using Database Triggers</strong></td>
<td>Describes database triggers and how you can use them with Splice Machine.</td>
</tr>
<tr>
<td><strong>Using Foreign Keys</strong></td>
<td>Describes our implementation of foreign keys and how our implementation ensures referential integrity.</td>
</tr>
<tr>
<td><strong>Using Window Functions</strong></td>
<td>A quick summary of window functions, as implemented in Splice Machine SQL.</td>
</tr>
<tr>
<td><strong>Using Temporary Tables</strong></td>
<td>Describes how to use temporary tables with Splice Machine.</td>
</tr>
<tr>
<td><strong>Using the VTI Interface</strong></td>
<td>Allows you to use an SQL interface with data that is external to your database.</td>
</tr>
<tr>
<td><strong>Using External Tables</strong></td>
<td>Describes the use of external tables (which are stored as flat files outside of your database) with Splice Machine.</td>
</tr>
<tr>
<td><strong>Compaction and Vacuuming</strong></td>
<td>Describes how compaction and vacuuming are used to improve database performance.</td>
</tr>
</tbody>
</table>

The developer fundamentals topics that apply only to our on-premise-database product are in a separate subsection of this section.
Using Compaction and Vacuuming

Compaction and vacuum are two mechanisms for clearing the physical space occupied by previously deleted items in your database.

About Compaction

To understand compaction, you need a basic understanding of how HBase stores information:

- Each HBase table has one or more Regions.
- Each HBase table has one or more column families.
- Each region and column family has a store that contains:
  - A buffer, MemStore, that holds in-memory modifications until it is flushed to store files. There is one MemStore per region and column family.
  - Store Files, which are also called HFiles, that are created when Memstore fills it.

Every time HBaseflushes a Memstore, it creates a new and immutable store file.

When you delete rows (tuples) from your database, Splice Machine marks the rows as deleted, but the physical space continues to be used until the table is compacted. As the physical space fills, reading live data from a table can require extra disk seeks, slowing performance. Compaction manages this problem.

Minor and Major Compaction

There are two kinds of compaction:

- Minor compactions happen automatically and continuously, as needed: HBase initiates this action when a region's space is getting overly full. Minor compactions combine a number of smaller HFiles into one larger HFile; this improves performance by reducing the number of disk reads required to read a row from a table. HBase runs minor compactions when the a store (region and column family) reaches the number of HFiles specified in the hbase.hstore.compactionThreshold property value.

- A Major compaction actually reads every block of data from the every store file in a Region, and rewrites only the live data to a single store file. This permanently deletes the rows that were previously marked as deleted. HBase runs major compactions on a scheduled interval, which is specified in the hbase.hregion.majorcompaction property; the default value for this property in Splice Machine is 7 days. Note that major compactions are very resource intensive and time consuming.

You can also optionally run a major compaction on a schema or table manually:

- You use the SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA procedure to run a major compaction on an entire schema. For example, if you've imported an entire database, you should seriously consider running a major compaction on the schema, because loading a lot of data can generate a large number of small store files; a major compaction will result in well-arranged regions that are roughly 50% full, which leaves room for growth and change.
You use the `SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE` procedure to run a major compaction on a specific table. For example, if you've imported a large dataset into a table or deleted a large number of rows from a table in your database, you may want to compact that table.

**Vacuuming**

When you drop a table from your database, Splice Machine marks the space occupied by the table as *deleted*, but does not actually free the physical space. That space is only reclaimed when you call the `SYCS_UTIL.VACUUM` system procedure, which does the following:

1. Waits for all previous transactions to complete (and times out if this takes too long).
2. Gets a list of all of the HBase tables in use.
3. Compares that list with a list of objects currently in use in your database, and deletes any HBase tables that are no longer in use in your database.

This is a synchronous operation; when it completes, you'll see the following message:

```
Ready to accept connections.
```
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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomicity</td>
<td>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.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Consistency</td>
<td>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.</td>
</tr>
<tr>
<td>Isolation</td>
<td>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.</td>
</tr>
<tr>
<td>Durability</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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.

**Additional Information About Snapshot Isolation**


- Our implementation does not have the Phantom Reads anomaly, when you apply the strict interpretation of that anomaly: in Splice Machine, if you execute the same query twice on the same transaction, you will get the same results.

- Splice Machine does not, however, protect against one form of this anomaly in its broadest interpretation; this has to do with how additions or modifications of rows in a predicate interact between transactions. This is described in the MVCC section of this document.

The diagram and table (Table 4) in the summary section of this paper, which summarize the different isolation levels and which anomalies they allow or disallow. Splice Machine allows: Write skew and a specific type of phantom reads (as noted above), but not phantom reads in general.

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.

When a transaction fails due to a write-write conflict, your application must restart the transaction; Splice Machine does not automatically restart such transactions.

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.

NOTE: In this example, the T3’ transaction has to be retried by the user.
Here's a tabular version of the same transactional timeline, showing the values committed in the database over time, with added commentary:

<table>
<thead>
<tr>
<th>Time</th>
<th>Committed Values</th>
<th>Transactions</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>t1</td>
<td>10</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>Time</td>
<td>Committed Values</td>
<td>Transactions</td>
<td>Comments</td>
</tr>
<tr>
<td>------</td>
<td>------------------</td>
<td>--------------</td>
<td>---------</td>
</tr>
<tr>
<td>$t_2$</td>
<td></td>
<td>$T_1 \text{ Start}$</td>
<td>T1 starts. The starting values within its transactional context are: $A=10$, $B=20$, $C=0$.</td>
</tr>
<tr>
<td>$t_3$</td>
<td></td>
<td>$A=A+10$ [A=20]</td>
<td>T1 modifies the value of $A$ within its context.</td>
</tr>
<tr>
<td>$t_4$</td>
<td></td>
<td>$T_2 \text{ Start}$</td>
<td>T2 starts. The starting values within its transactional context are the same as for $T_1$: $A=10$, $B=20$, $C=0$.</td>
</tr>
<tr>
<td>$t_5$</td>
<td></td>
<td>$A=A+10$ [A=30]</td>
<td>T1 again modifies the value of $A$ within its context.</td>
</tr>
<tr>
<td>$t_6$</td>
<td>30 20 0</td>
<td>$\text{Commit}$</td>
<td>T1 commits its modifications to the database.</td>
</tr>
<tr>
<td>$t_7$</td>
<td></td>
<td>$T_3 \text{ Start}$</td>
<td>T3 starts. The starting values within its transactional context include the commits from $T_1$: $A=30$, $B=20$, $C=0$.</td>
</tr>
<tr>
<td>$t_8$</td>
<td></td>
<td>$T_1 \text{ End}$</td>
<td></td>
</tr>
<tr>
<td>$t_9$</td>
<td></td>
<td>$B=B+10$ [B=30]</td>
<td>T2 modifies the value of $B$ within its context.</td>
</tr>
<tr>
<td>$t_{10}$</td>
<td></td>
<td>$C=A+10$ [C=20]</td>
<td>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 $T_2$.</td>
</tr>
<tr>
<td>$t_{11}$</td>
<td>30 30 20</td>
<td>$\text{Commit}$</td>
<td>T2 commits its changes.</td>
</tr>
<tr>
<td>$t_{12}$</td>
<td></td>
<td>$T_2 \text{ End}$</td>
<td></td>
</tr>
<tr>
<td>$t_{13}$</td>
<td></td>
<td>$B=B+10$ [B=30]</td>
<td>T3 modifies $B$; since its context includes the value of $B$ before $T_2$ committed, it modifies the original value of $B$ [B=20] in its own context.</td>
</tr>
</tbody>
</table>
The Isolation Guarantee

The Isolation guarantee makes sure that the resulting state of the database is consistent with a serial execution of the transactions that were completed even if they ran concurrently. Note, however, that this does not mean that you'll see the same results from all possible concurrent executions of transactions, even when they all complete successfully. What it does mean is this:

For a given start state $S$, if you run transactions $T_1$, $T_2$, and $T_3$ concurrently, then the state at which you end up, $S'$, will be equivalent to the state generated by a serial execution of those transactions.

Those three transactions can execute in a number of different orders, yielding a different $S'$. For example, consider a starting state $S$ in which $row\; A = 10$, and we'll be running these three transactions:
<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>A += 10</td>
</tr>
<tr>
<td>T2</td>
<td>A = A*2</td>
</tr>
<tr>
<td>T3</td>
<td>A += 30</td>
</tr>
</tbody>
</table>

When you run these transactions concurrently, the possible resulting states include these:

<table>
<thead>
<tr>
<th>Transaction Sequence</th>
<th>S’ Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>S + T1 + T3 + T2</td>
<td>A = 100</td>
<td></td>
</tr>
<tr>
<td>S + T3 + T2 + T1</td>
<td>A = 90</td>
<td></td>
</tr>
<tr>
<td>S + T3</td>
<td>A = 40</td>
<td>T1 and T2 were aborted due to Write-Write conflicts</td>
</tr>
<tr>
<td>S + T1 + T2</td>
<td>A = 40</td>
<td>T3 was aborted due to Write-Write conflicts</td>
</tr>
</tbody>
</table>

As you can see, even when all transactions complete successfully, the end result depends on the order in which the transactions committed. And all of these results meet the Isolation guarantee. Splice Machine will enforce that guarantee by aborting a transaction when it detects a write-write conflict.

The Isolation guarantee would be violated by generating an ending state that *could not be generated by serially running the transactions*. For example, this sequence of actions is not valid because there is no serial sequence that could generate an end result of \( A = 20 \) if all transactions successfully commit.

1. T1 reads \( A = 10 \)
2. T2 reads \( A = 10 \)
3. T3 reads \( A = 10 \)
4. T3 writes \( A = 10 + 30 = 40 \)
5. T2 writes \( A = 10 * 2 = 20 \)
6. T1 writes \( A = 10 + 10 = 20 \)
7. End result is \( A = 20 \)

**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:

<table>
<thead>
<tr>
<th>Autocommit Status</th>
<th>How Changes Are Committed and Rolled Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Changes are automatically committed whenever the operation completes successfully. If an operation reports any error, the changes are automatically rolled back.</td>
</tr>
<tr>
<td>Disabled</td>
<td>Changes are only committed when you explicitly issue a commit command. Changes are rolled back when you explicitly issue a rollback command.</td>
</tr>
</tbody>
</table>

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> 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
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:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triggering event (or statement)</td>
<td>The SQL statement that causes a trigger to be fired. This can be one of the following types of statement:</td>
</tr>
<tr>
<td></td>
<td>- INSERT</td>
</tr>
<tr>
<td></td>
<td>- UPDATE</td>
</tr>
<tr>
<td></td>
<td>- DELETE</td>
</tr>
<tr>
<td>Trigger action</td>
<td>The procedure that contains the SQL statements to be executed when a triggering statement is issued and any trigger restrictions evaluate to TRUE.</td>
</tr>
<tr>
<td></td>
<td>A trigger action is one of the following:</td>
</tr>
<tr>
<td></td>
<td>- arbitrary SQL</td>
</tr>
<tr>
<td></td>
<td>- a call to a user-defined stored procedure</td>
</tr>
</tbody>
</table>
When a Trigger Fires
You can define both statement and row triggers as either before triggers or after triggers:

<table>
<thead>
<tr>
<th>Trigger Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Triggers</td>
<td>A before trigger fires before the statement's changes are applied and before any constraints have been applied.</td>
</tr>
<tr>
<td>After Triggers</td>
<td>An after trigger fires after all constraints have been satisfied and after the changes have been applied to the target table.</td>
</tr>
</tbody>
</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.

<table>
<thead>
<tr>
<th>Trigger Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement Triggers</td>
<td>A statement trigger fires once per triggering event, regardless of how many rows (including zero rows) are modified by the event.</td>
</tr>
<tr>
<td>Row Triggers</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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
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 or Avro 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 (or set of PARQUET files in a directory) stored on your computer:

```sql
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 data files in the directory 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.

External Table Schema Evolution for PARQUET Data Files

If your external data is stored in PARQUET format, you can specify the `MERGE SCHEMA` option when creating your external table. This is useful when your data has schema evolution: ordinarily (without the option), Splice Machine infers the schema for your data by examining the column data types of one of the data files in the directory of files specified in the `LOCATION` parameter. This works fine except when the schema of the external data changes; for schema evolution cases, you can specify `MERGE SCHEMA` to tell Splice Machine to infer the schema from all of the data files.

To use schema merging with the previous example, simply add the `MERGE SCHEMA` clause:

```sql
splice> CREATE EXTERNAL TABLE myExtTbl (
  col1 INT, col2 VARCHAR(24))
  PARTITIONED BY (col1)
  STORED AS PARQUET
  LOCATION '/users/myname/myParquetFile'
  MERGE SCHEMA;

0 rows inserted/updated/deleted
```

Merging schemas from a large set of external files is expensive in terms of performance, so you should only use this option when necessary.

This option is only available for data in PARQUET format, due to Spark restrictions.
Accessing and Updating an ORC File

The following statement creates an external table for the ORC file stored in an AWS S3 bucket and inserts data into it.

NOTE: See the Configuring an S3 Bucket for Splice Machine Access topic for information about accessing data on S3.

```
splice> CREATE EXTERNAL TABLE myExtTbl2
       (col1 INT, col2 VARCHAR(24))
       PARTITIONED BY (col1)
       STORED AS ORC
       LOCATION 's3a://myOrcData/myName/myOrcFile';
0 rows inserted/updated/deleted
splice> INSERT INTO myExtTbl2 VALUES (1, 'One'), (2, 'Two'), (3, 'Three');
3 rows inserted/updated/deleted
> 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 'hdfs:///tmp/myTables/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:

```sql
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
```

**AVRO Limitations**

Our current implementation of AVRO external tables has these limitations:

"It does not support the use of DECIMAL data values. This restriction will be eliminated in a future release."

"It does not support compressed AVRO files."

**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 `SYCS_UTIL.SYCS_REFRESHEXTERNAL_TABLE` built-in system procedure. For example:

```sql
splice> CALL SYCS_UTIL.SYCS_REFRESHEXTERNAL_TABLE('APP', 'myExtTable');
Statement executed.
```

**See Also**

The `CREATE EXTERNAL TABLE` statement.

The `SYCS_UTIL.SYCS_REFRESHEXTERNAL_TABLE` built-in system procedure.
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 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
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:

```java
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:

```java
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:

```java
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:

```java
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:

```java
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:

```java
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:

```sql
CREATE PROCEDURE getStatementStatistics(statement varchar(1024))
  PARAMETER STYLE JAVA
  LANGUAGE JAVA
  READS SQL DATA
  DYNAMIC RESULT SETS 1
  EXTERNAL NAME 'com.splicemachine.example.SparkStatistics.getStatementStatistics';
```

3. **Create a table to use**

Let's create a very simple table to illustrate use of our procedure:

```sql
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:

```sql
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.execute.ExecRow;
import com.splicemachine.db.iapi.sql.execute.ResultColumnDescriptor;
import com.splicemachine.db.iapi.sql.execute.ValueRow;
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.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) 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
        JavaRDDLocatedRow< 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));
    }
}

private static MultivariateStatisticalSummary getColumnStatisticsSummary(JavaRDD LocatedRow< resultSetRDD, int[] fieldsToConvert) throws StandardException{
    JavaRDDVector< vectorJavaRDD = SparkMLibUtils.locatedRowRDDToVectorRDD(resultSetRDD, fieldsToConvert);
    MultivariateStatisticalSummary summary = Statistics.colStats(vectorJavaRDD.rdd());
    return summary;
}

/*
 * Convert a ResultSet to JavaRDD
 */
private static JavaRDDLocatedRow< ResultSetToRDD (ResultSet resultSet) throws StandardException{
    EmbedResultSet40 ers = (EmbedResultSet40)resultSet;
    com.splicemachine.db.iapi.sql.ResultSet rs = ers.getUnderlyingResultSet();
    JavaRDDLocatedRow< resultSetRDD = SparkMLibUtils.resultSetToRDD(rs);
    return resultSetRDD;
}
```java
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.getNumNonzeros().toArray();
        double[] variance = summary.variance().toArray();
        double[] normL1 = summary.normL1().toArray();
        double[] normL2 = summary.normL2().toArray();
        long count = summary.getCount();
        for (int i = 0; i < fieldsToConvert.length; ++i) {
            int columnIndex = fieldsToConvert[i];
            String columnName = metaData.getColumnName(columnIndex);
            ExecRow row = new ValueRow(9);
            row.setColumn(1, new SQLVarChar(columnName));
            row.setColumn(2, new SQLDouble(min[columnIndex - 1]));
            row.setColumn(3, new SQLDouble(max[columnIndex - 1]));
            row.setColumn(4, new SQLDouble(nonZeros[columnIndex - 1]));
            row.setColumn(5, new SQLDouble(variance[columnIndex - 1]));
            row.setColumn(6, new SQLDouble(normL1[columnIndex - 1]));
            row.setColumn(7, new SQLDouble(normL2[columnIndex - 1]));
            row.setColumn(8, new SQLLongInt(count));
            rows.add(row);
        }
        return rows;
    }
```
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](https://spark.apache.org/docs)
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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| CREATE TEMPORARY TABLE | CREATE [LOCAL|GLOBAL] TEMPORARY TABLE table-Name
{ ( {column-definition | Table-level constraint} 
  , {column-definition} ] * )
 ( column-name [ , column-name ] * )
} [NOLOGGING | ON COMMIT PRESERVE ROWS]; |
### 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
- 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

```sql
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 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:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Implemented by</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpliceFileVTI</td>
<td>For querying delimited flat files, such as CSV files.</td>
<td>com.splicemachine.derby.vti.SpliceFileVTI</td>
</tr>
<tr>
<td>SpliceJDBCVTI</td>
<td>For querying data from external sources that support the JDBC API.</td>
<td>com.splicemachine.derby.vti.SpliceJDBCVTI</td>
</tr>
</tbody>
</table>

Each of these classes implements the DatasetProvider interface, which is used by Spark for creating execution trees, and the VTICosting 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:
CREATE TABLE emp(
    name VARCHAR(10),
    title VARCHAR(30),
    age INT,
    something VARCHAR(12),
    date_hired TIMESTAMP,
    clock TIME
) AS
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);

<table>
<thead>
<tr>
<th>NAME</th>
<th>TITLE</th>
<th>AGE</th>
<th>SOMETHING</th>
<th>DATE_HIRED</th>
<th>CLOCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>sculligan</td>
<td>Relief Pitcher</td>
<td>27</td>
<td>08X-27-2015</td>
<td>2015-08-27 08:08:08.0</td>
<td>06:08:08</td>
</tr>
<tr>
<td>jpeepers</td>
<td>Catcher</td>
<td>37</td>
<td>08-26-2015</td>
<td>2015-08-21 08:09:08.0</td>
<td>08:08:08</td>
</tr>
<tr>
<td>mbamburger</td>
<td>Manager</td>
<td>47</td>
<td>08-25-2015</td>
<td>2015-08-20 08:10:08.0</td>
<td>10:08:08</td>
</tr>
<tr>
<td>gbrown</td>
<td>Batting Coach</td>
<td>46</td>
<td>08-24-2015</td>
<td>2015-08-21 08:11:08.0</td>
<td>11:08:08</td>
</tr>
<tr>
<td>jardson</td>
<td>Left Fielder</td>
<td>34</td>
<td>08-23X-2015</td>
<td>2015-08-22 08:12:08.0</td>
<td>11:08:08</td>
</tr>
</tbody>
</table>

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);

<table>
<thead>
<tr>
<th>NAME</th>
<th>TITLE</th>
<th>AGE</th>
<th>SOMETHING</th>
<th>DATE_HIRED</th>
<th>CLOCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>sculligan</td>
<td>Relief Pitcher</td>
<td>27</td>
<td>08X-27-2015</td>
<td>2015-08-27 08:08:08.0</td>
<td>06:08:08</td>
</tr>
<tr>
<td>jpeepers</td>
<td>Catcher</td>
<td>37</td>
<td>08-26-2015</td>
<td>2015-08-21 08:09:08.0</td>
<td>08:08:08</td>
</tr>
<tr>
<td>mbamburger</td>
<td>Manager</td>
<td>47</td>
<td>08-25-2015</td>
<td>2015-08-20 08:10:08.0</td>
<td>10:08:08</td>
</tr>
<tr>
<td>gbrown</td>
<td>Batting Coach</td>
<td>46</td>
<td>08-24-2015</td>
<td>2015-08-21 08:11:08.0</td>
<td>11:08:08</td>
</tr>
<tr>
<td>jardson</td>
<td>Left Fielder</td>
<td>34</td>
<td>08-23X-2015</td>
<td>2015-08-22 08:12:08.0</td>
<td>11:08:08</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatasetProvider</td>
<td>Used by Spark to construct execution trees.</td>
</tr>
<tr>
<td>VTICosting</td>
<td>Used by the Splice Machine optimizer to estimate the cost of operations.</td>
</tr>
</tbody>
</table>

Here’s the declaration:

```java
public class PropertiesFileVTI implements DatasetProvider, VTI_costing {

    //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:

```java
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:

```java
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 VTIOperation 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, ExecRow 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.

  ```java
  public double getEstimatedCostPerInstantiation( VTIEnvironment vtiEnv) throws SQLException;
  ```

- `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.

  ```java
  public double getEstimatedCostPerInstantiation( VTIEnvironment vtiEnv) throws SQLException;
  ```
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.

```java
public double supportsMultipleInstantiations(VTIEnvironment vtiEnv) throws SQLException;
```

**NOTE:** The VTI Costing 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:

```java
@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).

  ```java
  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.

  ```java
  ResultSetMetaData getMetaData() throws SQLException;
  ```

Here is the implementation of these methods for our PropertiesFileVTI class:
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.getPropertiesFile';
```

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.

```java
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.

```java
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.
**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:

<table>
<thead>
<tr>
<th>OrderID</th>
<th>CustomerID</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>144</td>
<td>1</td>
<td>250</td>
</tr>
<tr>
<td>167</td>
<td>1</td>
<td>150</td>
</tr>
<tr>
<td>202</td>
<td>1</td>
<td>250</td>
</tr>
<tr>
<td>209</td>
<td>1</td>
<td>325</td>
</tr>
<tr>
<td>224</td>
<td>1</td>
<td>125</td>
</tr>
<tr>
<td>66</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>94</td>
<td>2</td>
<td>200</td>
</tr>
<tr>
<td>127</td>
<td>2</td>
<td>300</td>
</tr>
<tr>
<td>444</td>
<td>2</td>
<td>400</td>
</tr>
</tbody>
</table>

This query will find the first Order ID for each specified Customer ID in the above table:

```sql
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:
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:

<table>
<thead>
<tr>
<th>Terms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>window function</td>
<td>A function that operates on a set of rows and produces output for each row.</td>
</tr>
<tr>
<td>window partition</td>
<td>The grouping of rows within a table.</td>
</tr>
<tr>
<td></td>
<td>Note that window partitions retains the rows, unlike aggregates,</td>
</tr>
<tr>
<td>window ordering</td>
<td>The sequence of rows within each partition; this is the order in which the rows are passed to the window function for evaluation.</td>
</tr>
<tr>
<td>window frame</td>
<td>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 <em>row or range</em> clause.</td>
</tr>
<tr>
<td>OVER clause</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>See the <a href="#">The OVER Clause</a> section below for more information.</td>
</tr>
<tr>
<td>partitioning clause</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>See the <a href="#">The Partition Clause</a> section below for more information.</td>
</tr>
<tr>
<td>ordering clause</td>
<td>Defines the ordering of rows within each partition.</td>
</tr>
<tr>
<td></td>
<td>See the <a href="#">The Order Clause</a> section below for more information.</td>
</tr>
<tr>
<td>Terms</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>frame clause</td>
<td>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. See the The Frame Clause section below for examples and more information.</td>
</tr>
</tbody>
</table>

**The OVER Clause**

A window function alway contains an OVER clause, which determines how the rows of the query are divided, or partitioned, for processing by the window function.

```sql
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:

```sql
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);
```

<table>
<thead>
<tr>
<th>OrderID</th>
<th>CustomerID</th>
<th>Amount</th>
<th>AverageOrderAmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>1</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>144</td>
<td>1</td>
<td>250</td>
<td>200</td>
</tr>
<tr>
<td>167</td>
<td>1</td>
<td>150</td>
<td>200</td>
</tr>
<tr>
<td>202</td>
<td>1</td>
<td>250</td>
<td>200</td>
</tr>
<tr>
<td>209</td>
<td>1</td>
<td>325</td>
<td>200</td>
</tr>
<tr>
<td>224</td>
<td>1</td>
<td>125</td>
<td>200</td>
</tr>
<tr>
<td>66</td>
<td>2</td>
<td>100</td>
<td>250</td>
</tr>
<tr>
<td>94</td>
<td>2</td>
<td>200</td>
<td>250</td>
</tr>
<tr>
<td>127</td>
<td>2</td>
<td>300</td>
<td>250</td>
</tr>
<tr>
<td>444</td>
<td>2</td>
<td>400</td>
<td>250</td>
</tr>
</tbody>
</table>

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**

```sql
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:
<table>
<thead>
<tr>
<th>value  PRECEDING</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURRENT ROW</td>
</tr>
<tr>
<td>value  FOLLOWING</td>
</tr>
</tbody>
</table>

value
A non-negative integer value.

For RANGE mode, you can only specify:

<table>
<thead>
<tr>
<th>CURRENT ROW</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNBOUNDED FOLLOWING</td>
</tr>
</tbody>
</table>

frameEnd
Specifies the end of the frame. The default value is CURRENT ROW.

For ROWS mode, you can specify:

<table>
<thead>
<tr>
<th>value  PRECEDING</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURRENT ROW</td>
</tr>
<tr>
<td>value  FOLLOWING</td>
</tr>
<tr>
<td>UNBOUNDED FOLLOWING</td>
</tr>
</tbody>
</table>

value
A non-negative integer value.

For RANGE mode, you can only specify:

<table>
<thead>
<tr>
<th>CURRENT ROW</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNBOUNDED FOLLOWING</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Frame Clause Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recycled</td>
<td>BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING</td>
</tr>
<tr>
<td>Cumulative</td>
<td>BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW</td>
</tr>
</tbody>
</table>
# Examples

This is a simple example that doesn't use a frame clause:

1. **Rank each year within a player by the number of home runs hit by that player:**
   
   ```sql
   RANK() OVER (PARTITION BY playerId ORDER BY H desc);
   ```

Here are some examples of window functions using frame clauses:

1. **Compute the running sum of G for each player:**
   
   ```sql
   SUM(G) OVER (PARTITION BY playerId ORDER BY yearID
   RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW);
   ```

2. **Compute the career year:**
   
   ```sql
   YearID - min(YEARID) OVER (PARTITION BY playerId
   RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) + 1;
   ```

3. **Compute a rolling average of games by player:**
   
   ```sql
   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:

```sql
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:

```sql
SELECT playerID, rank() OVER (PARTITION BY playerID ORDER BY G) as player_rank FROM Batting
WHERE player_rank = 1;
```

Instead, you need to use a subquery:

```sql
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
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](https://www.red-gate.com/simple-talk/articles/window-functions) about window functions are probably the most straightforward and comprehensive descriptions of window functions.

- This [Oracle Technology Network article](https://docs.oracle.com/cd/E11882_01/appdev.112/e25519/windowfunctions.htm) provides an excellent technical introduction.

- This [PostgreSQL page](https://www.postgresql.org/docs/current/window.html) introduces their version of window functions and links to other pages.

- This [PostgreSQL wiki page](https://wiki.postgresql.org/wiki/Window_functions) about SQL windowing queries page provides a succinct explanation of why windowing functions are used, and includes several useful examples.

- The [Wikipedia SQL SELECT](https://en.wikipedia.org/wiki/Window_function) page contains descriptions of specific window functions and links to other pages.
Working With Date and Time Values

This topic provides an overview of working with dates in Splice Machine, in these sections:

- Date and Time Functions
- Date Arithmetic

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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURRENT_DATE</td>
<td>Returns the current date as a DATE value.</td>
</tr>
<tr>
<td>DATE</td>
<td>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.</td>
</tr>
<tr>
<td>DAY</td>
<td>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.</td>
</tr>
<tr>
<td>EXTRACT</td>
<td>Extracts various date and time components from a date expression.</td>
</tr>
<tr>
<td>LAST_DAY</td>
<td>Returns a DATE value representing the date of the last day of the month that contains the input date.</td>
</tr>
<tr>
<td>MONTH</td>
<td>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.</td>
</tr>
<tr>
<td>MONTH_BETWEEN</td>
<td>Returns a decimal number representing the number of months between two dates.</td>
</tr>
<tr>
<td>MONTHNAME</td>
<td>Returns the month name from a date expression.</td>
</tr>
<tr>
<td>NEXT_DAY</td>
<td>Returns the date of the next specified day of the week after a specified date.</td>
</tr>
<tr>
<td>NOW</td>
<td>Returns the current date and time as a TIMESTAMP value.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>QUARTER</td>
<td>Returns the quarter number (1-4) from a date expression.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Returns a timestamp value from a TIMESTAMP value, a string representation of a timestamp value, or a string of digits representing such a value.</td>
</tr>
<tr>
<td>TIMESTAMPADD</td>
<td>Adds the value of an interval to a TIMESTAMP value and returns the sum as a new timestamp.</td>
</tr>
<tr>
<td>TIMESTAMPDIFF</td>
<td>Finds the difference between two timestamps, in terms of the specified interval.</td>
</tr>
<tr>
<td>TO_CHAR</td>
<td>Returns string formed from a DATE value, using a format specification.</td>
</tr>
<tr>
<td>TO_DATE</td>
<td>Returns a DATE value formed from an input string representation, using a format specification.</td>
</tr>
<tr>
<td>WEEK</td>
<td>Returns the week number (1-53) from a date expression.</td>
</tr>
<tr>
<td>YEAR</td>
<td>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.</td>
</tr>
</tbody>
</table>

Splice Machine displays TIME and TIMESTAMP values using the current time zone for the server to which you are connected.

**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**

```sql
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**

```sql
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` type
- `DATE` function
- `DAY`
- `EXTRACT`
- `LASTDAY`
- `MONTH`
- `MONTH_BETWEEN`
- `MONTHNAME`
- `NEXTDAY`
- `NOW`
- `QUARTER`
- `TIME` type
- `TIME` function
- `TIMESTAMP` type
- `TIMESTAMP` function
- `TO_CHAR`
- `TO_DATE`
- `WEEK`
Indexing in Splice Machine

This section includes these tutorial topics:

- Indexing Large Tables
Creating Indexes for Large Tables

Splice Machine implements indexes as tables, which means that creating a regular index is similar to creating a new table. This means that parallelization, which is essential for performance with big tables, is only possible when the index uses a sufficient number of regions. You can increase parallelism by optimizing how the index is split into regions.

Note that the standard form of the `CREATE INDEX` statement generates performant indexes for most tables: Splice Machine traverses the table and copies the specified column values to the index. For very large tables, this process can require a lot of time, and can generate an index that is split into uneven regions, which slows performance.

The remainder of this topic describes using the optional `SPLITKEYS` clause of our `CREATE INDEX` statement to efficiently create highly performant indexes for very large tables; this clause allows you to specify how to split the index into regions when it is generated, in these sections:

- Specifying Split Keys for Index Creation
- Using Bulk HFile Loading and Split Keys

Specifying Split Keys for Index Creation

There are several ways that you can specify split keys for creating your index, each of which is explored in this section:

- Specifying Split Keys with Automatic Sampling
- Specifying Split Keys in a File

Specifying Split Keys with Automatic Sampling

The simplest way to use split key indexing is to allow Splice Machine to sample the data in your table and determine the splits based on statistical analysis of that data by specifying `AUTO SPLITKEYS` in your `CREATE INDEX` statement. For example:

```
CREATE INDEX l_part_idx ON lineitem(
    l_partkey,
    l_orderkey,
    l_suppkey,
    l_shipdate,
    l_extendedprice,
    l_discount,
    l_quantity,
    l_shipmode,
    l_shipinstruct
) AUTO SPLITKEYS SAMPLE FRACTION 0.001;
```

Splice Machine scans and samples the base table to figure out how large the data range is. The sampled data set is used to encode index column values and calculate total size and quantile statistics. Then, using the total size, the maximum region size, and the sampling rate, we calculate the required number of regions required to accommodate the index data. Our calculation assumes that the regions will grow, so we only load a region half full.
You can optionally specify the sampling rate, as shown in the above example. If you leave the \texttt{SAMPLE FRACTION} subclause out, the default sampling rate is used:

\begin{itemize}
  \item The default sampling rates is specified in the \texttt{splice.bulkImport.sample.fraction} configuration property in your \texttt{split-site.xml} configuration file.
  \item The initial, default value in the configuration file is \texttt{0.005} (0.5%).
\end{itemize}

\textbf{How Auto Sampling Works}

The Splice Machine auto sampling algorithm calculates the number of regions required for the index with this formula:

\[
\text{num_regions} = \frac{\text{index\_size}}{\text{sampling\_rate}/\text{max\_region\_size}/2}
\]

The value of \texttt{max\_region\_size} is defined in the \texttt{hbase.hregion.max.filesize} property value.

Split keys are calculated according to the quantile statistics for the index and the number of regions; this automatically accounts for highly skewed data.

\textbf{Specifying Split Keys in a File}

Splice Machine encourages you to use automatic splitting; however, if you have a very large table and you're an expert user who is comfortable with manually determining split keys for the table, you can follow the steps in this section to create your own split keys CSV file and specify and specify it in your \texttt{CREATE INDEX} statement. by specifying \texttt{AUTO SPLITKEYS} in your \texttt{CREATE INDEX} statement

Manual computation of split keys is only appropriate if you know the size and shape of your data. For example, if you have a very large table that happens to have a roughly equal amount of daily data stretching for many months or years, it is relatively straightforward to estimate the number of regions to create for the table with minimal skewing. If you cannot answer questions about the ‘skewness’ of your data, you should probably use our automatic method to create your index split keys.

You can specify either \textit{logical} or \textit{physical keys}:

\begin{itemize}
  \item Logical keys are the primary key column values that you want to define the splits.
  \item Physical keys are actual split keys for the HBase table, in encoded HBase format.
\end{itemize}

\textbf{Example: Specifying Logical Split Keys in a File}

This is a step-by-step example of using \texttt{CREATE INDEX} with logical keys:

\begin{itemize}
  \item 1. Find index values that can horizontally split the index into roughly equal-sized paritions. Here is some guidance for calculating the split keys:
    \begin{itemize}
      \item a. First, estimate the number of regions required for the index by inspecting how many regions are used for the base table. The index usually needs less regions to accommodate data, because index data only contains index and primary key values. You can estimate from the table definition to
    \end{itemize}
\end{itemize}
calculate the number of regions:

\[
\text{num\_regions} = \frac{\text{index\_column\_size} + \text{PK\_column\_size}}{\text{total\_column\_size}}
\]

b. Run an SQL query to calculate some statistics and generate a histogram for leading index columns. For example, assuming the first column is `index\_col1`, we can use this statement:

\[
\text{splice> SELECT index\_col1, COUNT(*) FROM TABLE GROUP BY index\_col1 ORDER BY 1;}
\]

c. Given the estimated number of regions and the histogram, you can find keys that can split the index into roughly equal sized regions.

2. Next, calculate index values that will create evenly-sized regions for the index, and specify them in a CSV file. Since the example we're using includes 9 index columns, each line in our CSV file specifies values for those columns that define a split. For example, here's a partial listing of such a file:

<table>
<thead>
<tr>
<th>Split Key</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>6952</td>
<td>67431, 6953, 1998-01-29, 70640.1, 0.06, 38, SHIP, COLLECT COD</td>
</tr>
<tr>
<td>13491</td>
<td>2563013, 5993, 1997-05-19, 12640.41, 0.09, 9, AIR, NONE</td>
</tr>
<tr>
<td>20158</td>
<td>3540711, 7665, 1994-07-29, 5390.75, 0.07, 5, SHIP, COLLECT COD</td>
</tr>
<tr>
<td>26665</td>
<td>527524, 6666, 1994-12-10, 54116.44, 0.06, 34, SHIP, COLLECT COD</td>
</tr>
<tr>
<td>32488</td>
<td>1364581, 2489, 1996-08-15, 15625.28, 0.05, 11, FOB, TAKE BACK RETURN</td>
</tr>
</tbody>
</table>

3. Save the split keys in a CSV file on HDFS. For this example, we create a local CSV file `/tmp/l_part_idx.csv`, and then copy it to HDFS with this command:

```
hadoop dfs -copyFromLocal /tmp/l_part_idx.csv /tmp/l_part_idx.csv
```

4. Create the index with the CREATE INDEX statement. For example:

```
CREATE INDEX l_part_idx ON lineitem(
  l_partkey,
  l_orderkey,
  l_suppkey,
  l_shipdate,
  l_extendedprice,
  l_discount,
  l_quantity,
  l_shipmode,
  l_shipinstruct
) LOGICAL SPLITKEYS LOCATION '/tmp/l_part_idx.csv';
```

**Example: Specifying Physical Split Keys in a File**

This is a variation of the same step-by-step example that uses physical keys instead of logical keys:
1. Find index values that can horizontally split the index into roughly equal-sized paritions. Here is some guidance for calculating the split keys:
   
   a. First, estimate the number of regions required for the index by inspecting how many regions are used for the base table. The index usually needs less regions to accommodate data, because index data only contains index and primary key values. You can estimate from the table definition to calculate the number of regions:
      
      \[ \text{num\_regions} = \frac{\text{index\_column\_size} + \text{PK\_column\_size}}{\text{total\_column\_size}} \]
   
   b. Run an SQL query to calculate some statistics and generate a histogram for leading index columns. For example, assuming the first column is `index\_col1`, we can use this statement:
      
      \[
      \text{splice> SELECT index\_col1, COUNT(*) FROM TABLE GROUP BY index\_col1 ORDER BY 1;}
      \]
   
   c. Given the estimated number of regions and the histogram, you can find keys that can split the index into roughly equal sized regions.

2. Next, calculate index values that will create evenly-sized regions for the index, and supply the encoded HBase table values in a file. Here’s a partial example of the contents of a text file that contains such values:

   \[
   \xDB(\x00\xE1\x07g\x00\xDB)\x00\x1F\xD4=\x00\xE4\x81u\x12\x01\x00\xDEp\x01\x00\x1E1\x01\x00UJKR""""""""\x00EQQNGE"EQF""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
   \]

3. Save the split keys in a file on HDFS.

4. Create the index with the `CREATE INDEX` statement. For example:
CREATE INDEX l_part_idx ON lineitem(
  l_partkey,
  l_orderkey,
  l_suppkey,
  l_shipdate,
  l_extendedprice,
  l_discount,
  l_quantity,
  l_shipmode,
  l_shipinstruct
) PHYSICAL SPLITKEYS LOCATION '/tmp/l_part_idx.txt' HFILE LOCATION '/tmp/hfile';

Using Bulk HFile Loading and Split Keys

You can use HBase Bulk HFile loading with split keys to increase the performance of index creation for a large table. You can use bulk HFiles no matter how you specify your split keys.

NOTE: Bulk HFile index creation does not perform unique constraint checking.

The examples are almost exactly the same as those in the previous section; the only difference is that each of these examples uses bulk HFiles to improve index creation performance, which means that you have to create a staging directory to temporarily store the generated HFiles and specify the location of that directory in the HFILE LOCATION subclause of your CREATE INDEX statement.

The temporary HFiles are moved to HBase once index creation is complete.

Example: Using Bulk HFile Loading with Sampled Split Keys

Here is a step-by-step example of using automatic splitting with Bulk HFiles to create an index on an existing table:

1. Create a staging directory on HDFS to store the generated HFiles; for example:
   ```
   sudo -su hdfs hadoop fs -mkdir hdfs:///tmp/hfile
   ```

2. Ensure that the staging directory has permissions that allow Splice Machine to write your HFiles to it.

   NOTE: For a system employing encrypted HDFS zones, make sure that the staging directory is in the same encryption zone as HBase.
3. Create the index with the `CREATE INDEX` statement. For example:

```sql
CREATE INDEX L_PART_IDX ON lineitem(
  l_partkey,
  l_orderkey,
  l_suppkey,
  l_shipdate,
  l_extendedprice,
  l_discount,
  l_quantity,
  l_shipmode,
  l_shipinstruct
) AUTO SPLITKEYS SAMPLE FRACTION 0.001 HFILE LOCATION '/tmp/hfile';
```

**Example: Using Bulk HFile Loading with Specified Split Keys**

Here is a step-by-step example of specifying your own split keys in a file and using them with Bulk HFile loading:

1. Create a staging directory on HDFS to store the generated HFiles; for example:
   ```bash
   sudo -su hdfs hadoop fs -mkdir hdfs:///tmp/hfile
   ```

2. Ensure that the staging directory has permissions that allow Splice Machine to write your HFiles to it.
   
   **NOTE:** For a system employing encrypted HDFS zones, make sure that the staging directory is in the same encryption zone as HBase.

3. Find index values that can horizontally split the index into roughly equal-sized partitions. Here is some guidance for calculating the split keys:

   a. First, estimate the number of regions required for the index by inspecting how many regions are used for the base table. The index usually needs less regions to accommodate data, because index data only contains index and primary key values. You can estimate from the table definition to calculate the number of regions:
      ```
      num_regions = index_column_size + PK_column_size) / total_column_size
      ```

   b. Run an SQL query to calculate some statistics and generate a histogram for leading index columns. For example, assuming the first column is `index_col1`, we can use this statement:
      ```sql
      splice> SELECT index_col1, COUNT(*) FROM TABLE GROUP BY index_col1 ORDER BY 1;
      ```

   c. Given the estimated number of regions and the histogram, you can find keys that can split the index
into roughly equal sized regions.

4. Next, calculate index values that will create evenly-sized regions for the index, and save them in a file. You can specify logical (primary key) values in a CSV file, or you can specify physical (HBase encoded) values in a text file.

For example, here's a partial listing of a CSV file:

```
6952,67431,6953,1998-01-29,70640.1,0.06,38,SHIP      ,COLLECT COD
13491,2563013,5993,1997-05-19,12640.41,0.09,9,AIR     ,NONE
20158,3540711,7665,1994-07-29,5390.75,0.07,5,SHIP      ,COLLECT COD
26665,527524,6666,1994-12-10,54116.44,0.06,34,SHIP     ,COLLECT COD
32488,1364581,2489,1996-08-15,15625.28,0.05,11,FOB    ,TAKE BACK RETURN
```

And here's a partial listing of a file with encoded HBase split keys:

```
\xDB(\x00\xE1\x07g\x00\xDB)\x00\x1F\x00\xE4\x81u\x12\x01\x00\xDEp\x01\x00\x1EI\x01\x00UJKR""""""""""\x00\xE4\x81u\x12\x01\x00\xDEp\x01\x00\x1EI\x01\x00UJKR""""""""""\x00\xE4\x81u\x12\x01\x00\xDEp\x01\x00\x1EI\x01\x00UJKR""""""""""\x00\xE4\x81u\x12\x01\x00\xDEp\x01\x00\x1EI\x01\x00UJKR""""""""""
```

5. Create or copy the file on HDFS. For example, we can create a local CSV file `/tmp/l_part_idx.csv`, and then copy it to HDFS with this command:

```
hadoop dfs -copyFromLocal /tmp/l_part_idx.csv /tmp/l_part_idx.csv
```

6. Create the index with the `CREATE INDEX` statement. For example:
CREATE INDEX l_part_idx ON lineitem(
  l_partkey,
  l_orderkey,
  l_suppkey,
  l_shipdate,
  l_extendedprice,
  l_discount,
  l_quantity,
  l_shipmode,
  l_shipinstruct
) LOGICAL SPLITKEYS LOCATION '/tmp/l_part_idx.csv' HFILE LOCATION '/tmp/hfile';

If you’re using a file with physical keys, the last line looks like this instead:

CREATE INDEX l_part_idx ON lineitem(
  l_partkey,
  l_orderkey,
  l_suppkey,
  l_shipdate,
  l_extendedprice,
  l_discount,
  l_quantity,
  l_shipmode,
  l_shipinstruct
) PHYSICAL SPLITKEYS LOCATION '/tmp/l_part_idx.txt' HFILE LOCATION '/tmp/hfile';

See Also

» CREATE INDEX

» Best Practices: Bulk Importing Flat Files

» Best Practices: Bulk Importing Flat Files
Supported Authentication Mechanisms

You can use any of the following authentication mechanisms with Splice Machine; click the mechanism name link to navigate to a topic page that describes how to configure and use the mechanism.

<table>
<thead>
<tr>
<th>Authentication Mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Any user ID and password combination is allowed to connect to database.</td>
</tr>
<tr>
<td>NATIVE</td>
<td>User IDs in a database table are validated against the corresponding, encrypted password. This is the default authentication setting for Splice Machine installations.</td>
</tr>
<tr>
<td>KERBEROS</td>
<td>User IDs are validated against a Kerberos server.</td>
</tr>
<tr>
<td>LDAP</td>
<td>User IDs are validated against an existing LDAP service.</td>
</tr>
</tbody>
</table>

**ENTERPRISE ONLY:** This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product; contact Splice Machine Sales for information.
Configuring Authentication on Your Platform

You can configure authentication for your Splice Machine database by adding or modifying properties in your HBase configuration file. The location of the configuration file you need to modify depends on which platform you’re using:

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

Specific property settings are listed on the topic page for each authentication mechanism, as listed in the previous section.

Disabling Authentication

To disable authentication for your Splice Machine database, set the splice.authentication property in your configuration file to NONE:

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

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

```xml
<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 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:

```xml
<property>
    <name>splice.authentication.native.create.credentials.database</name>
    <value>true</value>
</property>
```
Using LDAP Authentication

This topic describes how to use LDAP authentication in Splice Machine, in these subsections:

- About LDAP Authentication in Splice Machine
- LDAP Property Settings
- Authenticating With an LDAP Group
- Troubleshooting LDAP

About LDAP Authentication in Splice Machine

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 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:port-number</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.searchAuth.password</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>search-filter-criteria</value>
</property>

Notes about the LDAP property values:

- Specify both the location of your external LDAP server host and the port number in the `splice.authentication.ldap.server` property. The default openLDAP port is 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 (distinct name)).
  - 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.searchAuth.password` property specifies password Splice Machine should use to perform the DN search; this is the password of the DN specified in `ldap.searchAuthDN` property.

- 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. An example is: `(&(objectClass=*)(uid=%USERNAME%))`
**Authenticating With an LDAP Group**

To use an LDAP GROUP, you must create a Splice Machine database user for that group. You can then assign privileges to that user, and everyone belonging to the LDAP GROUP will gain those privileges.

For example, given an LDAP GROUP named test_devel:

```
splice> call syscs_util.syscs_create_user('test_devel', 'test_devel');
Statement executed.
splice> create schema test_devel_schema;
0 rows inserted/updated/deleted
splice> create role test_devel_role;
0 rows inserted/updated/deleted
splice> grant all privileges on schema test_devel_schema to test_devel_role;
0 rows inserted/updated/deleted
splice> grant cdl_devl_role to test_devel;
0 rows inserted/updated/deleted
```

You can now connect as user testuser, who belongs to the test_devel LDAP Group:

```
splice> connect 'jdbc:splice://localhost:1527/splicedb;user=testuser;password=testpwd';
splice> create table test_devel_schema.t1(a int);
0 rows inserted/updated/deleted
splice> insert into test_devel_schema.t1 values (10), (20), (30);
3 rows inserted/updated/deleted
splice> select * from test_devel_schema.t1;
A
---------
10
20
30
3 rows selected
```

**LDAP Groups and Splice Machine**

Given an LDAP user and its DN (*Distinct Name*), Splice Machine honors the LDAP groups that user belongs to from two sources:

- the first *CN* (*Common Name*) in the DN, which may or may not be the same as the LDAP user name
- the user’s *memberOf* property

Here’s an example for an LDAP user with these DN and *memberOf* attributes:

```
# user3, Users, splicemachine.colo
dn: cn=user3,ou=Users,dc=splicemachine,dc=colo
memberOf: cn=foo,ou=groups,dc=splicemachine,dc=colo
memberOf: cn=mygroup,ou=groups,dc=splicemachine,dc=color
```
Splice Machine treats `user3`, `foo`, and `mygroup` as the LDAP groups to which `user3` belongs. All privileges granted to those three groups are inherited by the LDAP user `user3`.

**LDAP Group Names and Splice Machine**

When using an LDAP Group name in a `GRANT` or `REVOKE` statement: if the group name contains characters other than alphanumerics or the underscore character (A-Z, a-z, 0-9, _), you must:

- Enclose the group name in double quotes
- Convert all alphabetic characters in the group name to uppercase.

For example, if you are granting rights to an LDAP Group with name `This-is-my-LDAP-Group`, you would use a statement like this:

```
GRANT SELECT ON TABLE Salaries TO "THIS-IS-MY-LDAP-GROUP";
```

**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
```

**Troubleshooting LDAP**

There is a known issue when authenticating with LDAP protocol to an Active Directory instance. If you see “Unprocessed Continuation Reference” error messages in the Splice Machine region server logs, this is typically caused by using a default Active Directory port (369/636). To fix:

- Change port 369 to port 3268 for LDAP
- Change port 636 to port 3269 for LDAPS.

Using the alternate port allows for a broader search and lets you follow references.

Secure ldap is always preferred since it is the only way to securely encrypt your users’ passwords.
Enabling Splice Machine Kerberos Authentication on Your Cluster

Kerberos authentication in Splice Machine uses an external KDC server.

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


Follow these steps to enable Kerberos authentication:

1. Use KDC to create a new principal and generate a keytab file. For example:
   ```
   # kadmin.local
   addprinc -randkey jdoe@yourdomain.com
   ```

2. Set the password for the new principal:
   ```
   # kadmin.local: cpw jdoe
   Enter password for principal "jdoe@yourdomain.com"
   ```

3. Create keytab file jdoe.keytab:
   ```
   # kadmin.local: xst -k /tmp/jdoe.keytab jdoe@yourdomain.com
   ```

4. Copy the keytab file to your region servers.

5. Verify that you can successfully kinit with the new keytab file and access the hadoop file system on the region server node:
   ```
   $ kinit jdoe@yourdomain.com -kt /tmp/jdoe.keytab
   ```

6. Configure kerberos authentication against the database by setting your authentication properties as follows:
   ```
   <property>
   <name>splice.authentication</name>
   <value>KERBEROS</value>
   </property>
   ```
On Cloudera Manager, you can go to HBase Configuration and search for splice.authentication. Change the value to KERBEROS for both Client Configuration and Service Configuration and restart HBase.

7. Grant privileges to the new user. For example, here we grant all privileges to user jdoe on a table named myTable:

```
splice> GRANT ALL PRIVILEGES ON Splice.myTable to jdoe;
```

**NOTE:** You can enable Kerberos mode on a Cloudera cluster using the configuration wizard described here: [https://www.cloudera.com/documentation/enterprise/5-8-x/topics/cm_sg_intro_kerb.html](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/cm_sg_intro_kerb.html)
## Splice Machine On-Premise-Only Fundamentals

This section contains the following fundamental developer topics that are only relevant to users of our On-Premise-Database product:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Describes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using HCatalog</td>
<td>How to use Splice Machine with HCatalog.</td>
</tr>
<tr>
<td>Using HAProxy with Splice Machine</td>
<td>How to configure HAproxy for use with Splice Machine.</td>
</tr>
<tr>
<td>Using MapReduce</td>
<td>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.</td>
</tr>
<tr>
<td>Working with HBase</td>
<td>Working in HBase with Splice Machine.</td>
</tr>
<tr>
<td>Using Zeppelin</td>
<td>Using Apache Zeppelin with your on-premise database.</td>
</tr>
</tbody>
</table>

The general developer fundamentals topics, which apply to all of our products, are in a separate subsection of this section.
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.

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**:
   ```bash
   # yum install haproxy
   ```

   **NOTE**: You may use a different haproxy package, depending on which Linux distribution you’re using.

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 example below uses the *least connections* load-balancing algorithm. There are other load balancing algorithms, such as round robin, that can also be used, depending on the nature of your desired workload distribution.

Here is the `haproxy.cfg` file for this example:
# Global settings

```markdown
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

```markdown
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 port
# so that HAProxy can balance between the splice engine cluster nodes
# where each node's splice engine instance is listening on port 1527

```markdown
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 msecs, 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:

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Instructions</th>
</tr>
</thead>
</table>
   | Redhat / CentOS EL6 | # chkconfig haproxy on  
   |                | # service haproxy start  
   |                | If you change the configuration file, reload it with this command:  
   |                | # service haproxy reload |
### Distribution

<table>
<thead>
<tr>
<th>Redhat / CentOS EL7</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td># systemctl enable haproxy</td>
</tr>
<tr>
<td>ln -s '/usr/lib/systemd/system/haproxy.service' '/etc/systemd/system/multi-user.target.wants/haproxy.service'</td>
</tr>
<tr>
<td># systemctl start haproxy</td>
</tr>
</tbody>
</table>

If you change the configuration file, reload it with this command:

# systemctl haproxy reload

**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=YourUserId;password=YourPassword
```

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:

```
```

You'll see a report that looks similar to this:
HAProxy

Statistics Report for pid 751 on stl-colo-10-1-0-236.splicemachine.colo

> General process information

pid = 751 (process #1, rdpcon = 1)
uptime = 04:30:37m00s
system limits: memmax = unlimited; ulimit -n = 65536
maxsock = 65536; maxconn = 2000; maxpipes = 0
current conn = 1027, current pipes = 65, conn rate = 1/sec
Running taskset: 1/1208, idle = 96 %

<table>
<thead>
<tr>
<th>Queue</th>
<th>Cur</th>
<th>Max</th>
<th>Limit</th>
<th>Cur</th>
<th>Max</th>
<th>Limit</th>
<th>Total</th>
<th>La/Tot</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontend</td>
<td>0</td>
<td>0</td>
<td>118</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>OPEN</td>
</tr>
<tr>
<td>srv127</td>
<td>0</td>
<td>0</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>250</td>
<td>250</td>
<td>751</td>
<td>751</td>
</tr>
<tr>
<td>srv128</td>
<td>0</td>
<td>0</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>250</td>
<td>250</td>
<td>750</td>
<td>750</td>
</tr>
<tr>
<td>srv130</td>
<td>0</td>
<td>0</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>250</td>
<td>250</td>
<td>750</td>
<td>750</td>
</tr>
<tr>
<td>Backend</td>
<td>0</td>
<td>0</td>
<td>118</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>1000</td>
<td>200</td>
<td>200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Queue</th>
<th>Cur</th>
<th>Max</th>
<th>Limit</th>
<th>Cur</th>
<th>Max</th>
<th>Limit</th>
<th>Total</th>
<th>La/Tot</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontend</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>200</td>
<td>0</td>
<td>85</td>
<td>61755</td>
<td>118907</td>
<td>0</td>
</tr>
<tr>
<td>Backend</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>200</td>
<td>71</td>
<td>61755</td>
<td>118907</td>
<td>0</td>
</tr>
</tbody>
</table>

Display option:
- Scope:
- Hide: DOWN 4
- Disable refresh
- Refresh rate
- CSV export

Note: "NOOLV"OILV" = UP with load-balancing disabled.
Using Splice Machine with HCatalog

Apache HCatalog is a metadata and table management system for the broader Hadoop platform. HCatalog’s 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.

This is an On-Premise-Only topic!  Learn more

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=yourUserId\;password=yourPassword", "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.
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.
Hive Integration on a Kerberized Cluster

To integrate with Hive on a Kerberized cluster, you need to:

1. Make the following configuration changes in the `hive-site.xml` configuration file:

   ```xml
   <property>
     <name>hive.server2.thrift.port</name>
     <value>10001</value>
   </property>
   
   <property>
     <name>hive.server2.authentication</name>
     <value>KERBEROS</value>
   </property>
   
   <property>
     <name>hive.server2.authentication.kerberos.principal</name>
     <value>hive/<HiveServer2Host></value> <!-- for example: my_hive/test@HADOOP.TEST.COM -->
   </property>
   
   <property>
     <name>hive.server2.authentication.kerberos.keytab</name>
     <value>/home/user/user.keytab</value>
   </property>
   ```

2. Connect to the Hive server with Hive principal.

3. Grant the Hive user privileges to access HBase:

   ```sql
   GRANT 'hive' 'RWCA';
   ```

4. Update your code to authenticate with the Hive principal/keytab for accessing HBase; alternatively you can authenticate with a Hive delegation token.

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 hctest(col1 int, col2 varchar(20));
splice> insert into hctest values(1, 'row1');
splice> insert into hctest values(2, 'row2');
splice> insert into hctest values(3, 'row3');
splice> insert into hctest values(4, 'row4');

2. Verify the table
Verify that hctest is set up correctly:

splice> describe hctest;

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>TYPE_NAME</th>
<th>DECIMAL</th>
<th>NUMERIC</th>
<th>COLUMN_DEFAULT</th>
<th>CHAR_OCTET_LENGTH</th>
<th>IS_NULLABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL1</td>
<td>INTEGER</td>
<td>0</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>YES</td>
</tr>
<tr>
<td>COL2</td>
<td>VARCHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>20</td>
<td>40</td>
<td>YES</td>
</tr>
</tbody>
</table>

2 rows selected
splice> select * from hctest;

<table>
<thead>
<tr>
<th>COL1</th>
<th>COL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>row1</td>
</tr>
<tr>
<td>2</td>
<td>row2</td>
</tr>
<tr>
<td>3</td>
<td>row3</td>
</tr>
<tr>
<td>4</td>
<td>row4</td>
</tr>
</tbody>
</table>

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=yourUserId\;password=yourPassword", "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:
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:

```sql
splice> describe tblA;
COLUMN_NAME | TYPE_NAME | DES| NUM | COLUMN | COLUMN_DEF | CHAR_OCTETS | IS_NULL
-------------|-----------|----|-----|--------|-------------|-------------|-------
-             |           |    |     |        |             |             |       
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.
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_OCTET|IS_NULL
--------------
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;u
ser=yourUserId\;password=yourPassword", "splice.tableName"="SPLICE.tbl
B");

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
Splice Machine Map Reduce API

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:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpliceJob</td>
<td>Creates a transaction for the MapReduce job.</td>
</tr>
<tr>
<td>SMInputFormat</td>
<td>Creates an object that:</td>
</tr>
<tr>
<td></td>
<td>» uses Splice Machine to scan the table and decode the data</td>
</tr>
<tr>
<td></td>
<td>» returns an ExecRow (typed data) object</td>
</tr>
<tr>
<td>SMOutputFormat</td>
<td>Creates an object that:</td>
</tr>
<tr>
<td></td>
<td>» writes to a buffered cache</td>
</tr>
<tr>
<td></td>
<td>» dumps the cache into Splice Machine</td>
</tr>
<tr>
<td></td>
<td>» returns an ExecRow (typed data) object.</td>
</tr>
<tr>
<td>SpliceMapReduceUtil</td>
<td>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.</td>
</tr>
</tbody>
</table>

**NOTE:** Each transaction must manage its own commit and rollback operations.

For information about and examples of using Splice Machine with HCatalog, see the <a href="Using Splice Machine with HCatalog topic.">Using Splice Machine with HCatalog topic.</a>
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:

```java
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:

```java
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:

```java
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.
   
   ```java
   job.commit();
   ```

   If the job fails, roll back the transaction.
   
   ```java
   job.rollback();
   ```
Connecting with Apache Zeppelin

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.

This is an On-Premise-Only topic! Zeppelin is already integrated and ready to use in our Database-as-Service product.

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Whatever name you like; we're using SpliceMachine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreter</td>
<td>Select jdbc from the drop-down list of interpreter types.</td>
</tr>
</tbody>
</table>
**default.url** | jdbc:splice:/myServer:1527/splicedb
--- | ---
(replace **myServer** with the name of the server that you're using)
**default password** | yourPassword
**default userId** | yourUserId
**common.max_count** | 1000
**default.driver** | com.splicemachine.db.jdbc.ClientDriver
**Artifacts** | Insert the path to the Splice Machine jar file; for example:

```
/tmp/db-client-2.5.0.1708-SNAPSHOT.jar
```

**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 bindings]

```
SpliceMachine [jdbc (default)]
spark     [spark]
spark pyspark [spark]
```

...
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:

```javascript
%jdbc %jdbc specif ies 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:

![Results of JDBC call](image)

```
<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>MIN</th>
<th>MAX</th>
<th>COUNT</th>
<th>NUM_NONZEROS</th>
<th>STANDARD_DEVIATION</th>
<th>MEAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGE</td>
<td>7</td>
<td>73</td>
<td>943</td>
<td>12</td>
<td>34</td>
<td></td>
</tr>
</tbody>
</table>
```

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.

```javascript
%jdbc %jdbcselect count(1) num_age, age from MOVIELENS.USER_DEMOGRAPHICS group by age;
```

The results of this paragraph:
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:
   
   a. Click the rightmost settings icon, then click **settings**.
   
   b. **Move age to the xAxis, and the number of people of that age to the yAxis.**
   
   c. You'll now see the distribution of ages:
d. Click the graphs button to select other data visualizations:

5. See the [Apache Zeppelin Documentation](#) to learn more about Zeppelin.
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:
### 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.

<table>
<thead>
<tr>
<th>TABLE_SCHEMA</th>
<th>TABLE_NAME</th>
<th>CONGLOM_ID</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>SYSALIASES</td>
<td>368</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSBACKUP</td>
<td>944</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSBACKUPITEMS</td>
<td>1056</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCHECKS</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCOLPERMS</td>
<td>640</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCOLUMNNS</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCOLUMNSTATS</td>
<td>1216</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCONGLOMERATES</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCONSTRAINTS</td>
<td>336</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSDEPENDS</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSFIELDS</td>
<td>288</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSFOREIGNKEYS</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSKEYS</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSPERMS</td>
<td>816</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSPHYSICALSTATS</td>
<td>1264</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSPRIMARYKEYS</td>
<td>1424</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSROLES</td>
<td>752</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSROUTINEPERMS</td>
<td>672</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSCHEMAS</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSEQUENCES</td>
<td>800</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSTATEMENTS</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLEPERMS</td>
<td>656</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLES</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTRIGGERS</td>
<td>304</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSUSERS</td>
<td>880</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSVIEWS</td>
<td>352</td>
<td></td>
</tr>
<tr>
<td>SYSIBM</td>
<td>SYSDUMMY1</td>
<td>1296</td>
<td></td>
</tr>
</tbody>
</table>
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:** Data is compressed in cluster installations of Splice Machine, but is not compressed in our standalone version.

**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.
Securing Your Splice Machine Database

This section includes the following topics related to securing your Splice Machine database and connections, including securing your JDBC connections and authorizing permissions for users and roles in your database:

- Securing Connections with SSL/TLS
- Accessing Splice Machine from Windows on a Kerberized Cluster
- Authorizing Users and Roles
- Summary of Permissions for Users and Roles
- Using Apache Ranger with Splice Machine
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 About 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.

Our Summary of Permissions for Users and Roles topic summarizes which permissions are available to and can be granted or revoked by different kinds of users and roles.

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. We strongly suggest changing these values.

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.

When you GRANT a role to a user, that role is automatically defined as a default role for that user, which means that whenever that user connects to the database, they will have the permissions associated with that role. This AS DEFAULT behavior is how GRANT operates by default. If you want to grant a user a role just for their current session, you can specify the NOT AS DEFAULT option in your GRANT statement.

NOTE: You can also revoke the association of a role as AS DEFAULT behavior for a user by using the NOT AS DEFAULT

See the GRANT statement documentation for examples of using AS DEFAULT and NOT AS DEFAULT with roles.

Using SET ROLE to Add Permissions
When a user connects to Splice Machine, that user is granted any permissions that are associated with the public user, and is granted any permissions associated with roles that have been granted by default to that user or the public user.

You can add additional roles for a user’s sessions with the SET ROLE statement, which adds a role for the current session.

To unset all roles for the user’s current session, you can call SET ROLE with an argument of NONE.

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
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, not 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 SSL/TLS Secure Connections

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

About Encrypted Connections

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:

```bash
% 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:

```bash
% keytool -export -alias MyServerName \
  -keystore ~/value/ServerKeyStore -rfc -file ServerCertificate \
  -storepass myPassword
Certificate stored in file <ServerCertificate>
% ls -ltr
```

Generate Client Certificates

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

```bash
% 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:
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
Certificate fingerprints:
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 favoriteServerCertificate \
  -file ServerCertificate -keyStore ./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
Certificate fingerprints:
    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
-Djavax.net.ssl.keyStore=/tmp/vault/ServerKeyStore
-Djavax.net.ssl.keyStorePassword=myPassword
-Djavax.net.ssl.trustStore=/tmp/vault/ServerTrustStore
-Djavax.net.ssl.trustStorePassword=secretServerTrustStorePassword

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:

<table>
<thead>
<tr>
<th>Security Mode</th>
<th>Connect Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>connect 'jdbc:splice://x.x.x.xxx:1527/splicedb;user=YourUserId;password=YourPassword';</td>
</tr>
<tr>
<td>Basic SSL</td>
<td>connect 'jdbc:splice://x.x.x.xxx:1527/splicedb;user=YourUserId;password=YourPassword;ssl=basic';</td>
</tr>
<tr>
<td>SSL w/Peer Authentication</td>
<td>connect 'jdbc:splice://x.x.x.xxx:1527/splicedb;user=YourUserId;password=YourPassword;ssl=peerAuthentication';</td>
</tr>
</tbody>
</table>

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/splicedb;user=YourUserId;password=YourPassword;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:
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:

```bash
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:

   ```bash
   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.net.ssl.trustStore.ssl.trustStorePassword=${CLIENT_SSL_TRUSTSTOREPASSWD}"
   
   -Djavax.net.ssl.trustStore.ssl.trustStorePassword=${CLIENT_SSL_TRUSTSTOREPASSWD}"
   
   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:

![Table of Sample Settings for New Interpreter]

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>common.max_count</td>
<td>1000</td>
</tr>
<tr>
<td>default.driver</td>
<td>com.splicemachine.db.jdbc.ClientDriver</td>
</tr>
<tr>
<td>default.password</td>
<td>admin</td>
</tr>
<tr>
<td>default.url</td>
<td>jdbc:spice://localhost:1527/splicodb;ssl=ServerAuth</td>
</tr>
<tr>
<td>default.user</td>
<td>spice</td>
</tr>
<tr>
<td>zeppelin.interpreter.localRepo</td>
<td>/Users/mepalai/zeppelin-0.7.2-bin-all/local-repo/2C94WH1P</td>
</tr>
<tr>
<td>zeppelin.interpreter.output.limit</td>
<td>102400</td>
</tr>
<tr>
<td>zeppelin.jdbc.auth.type</td>
<td></td>
</tr>
<tr>
<td>zeppelin.jdbc.concurrent.max_connection</td>
<td>10</td>
</tr>
<tr>
<td>zeppelin.jdbc.concurrent.use</td>
<td>true</td>
</tr>
<tr>
<td>zeppelin.jdbc.keytab.location</td>
<td></td>
</tr>
<tr>
<td>zeppelin.jdbc.principal</td>
<td></td>
</tr>
</tbody>
</table>

**Dependencies**

<table>
<thead>
<tr>
<th>artifact</th>
<th>exclude</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Users/mepalai/zeppelin-0.7.2-bin-all/local-repo/2C94WH1P</td>
<td></td>
</tr>
</tbody>
</table>

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.
Summary of Permissions for Users and Roles

This topic summarizes which permissions are available to and can be granted or revoked by the Splice user, regular users, and roles, in these sections:

- Permissions in System Schemas
- Permissions in the SPLICE Schema
- Permissions in Regular Schemas
- System Procedures and Routines Permissions

Summary of Permissions in System Schemas

The following table summarizes which permissions apply to and can be granted or revoked by the Splice user, regular users, and roles for tables in the system schemas:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Splice User</th>
<th>Regular User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Has Permission?</td>
<td>Can Grant or Revoke?</td>
<td>Has Permission?</td>
</tr>
<tr>
<td>Update/ Delete/ Insert</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Select</td>
<td>Yes</td>
<td>No</td>
<td>Yes, except for sys.sysusers</td>
</tr>
<tr>
<td>Create/Drop/ Alter table</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Create/Drop schema</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

These are the system schemas to which the above privileges apply:

- sys
- sysibm
- syscs_util
- syscs_diag
Permissions in the *SPLICE* Schema
The following table summarizes which permissions apply to and can be granted or revoked by the Splice user, regular users, and roles for tables in the SPLICE schema:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Splice User</th>
<th>Regular User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Has Permission?</td>
<td>Can Grant or Revoke?</td>
<td>Has Permission?</td>
</tr>
<tr>
<td>Update/ Delete/ Insert</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if granted the corresponding privilege.</td>
</tr>
<tr>
<td>Select</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if granted the corresponding privilege.</td>
</tr>
<tr>
<td>Create/ Drop/ Alter table ...</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if granted the corresponding privilege.</td>
</tr>
<tr>
<td>Create/ Drop schema</td>
<td>Yes</td>
<td>No</td>
<td>Yes for DROP SCHEMA, if the user becomes the owner of the SPLICE schema.</td>
</tr>
</tbody>
</table>

Permissions in Regular Schemas
The following table summarizes which permissions apply to and can be granted or revoked by the Splice user, regular users, and roles for tables in regular schemas:
### Permissions for System Procedures and Routines

The following table summarizes which permissions apply to and can be granted or revoked by the Splice user, regular users, and roles for system procedures and routines:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Splice User</th>
<th>Regular User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Has Permission?</td>
<td>Can Grant or Revoke?</td>
<td>Has Permission?</td>
</tr>
<tr>
<td>Update/Delete/Insert</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if schema owner, or if granted the corresponding privilege.</td>
</tr>
<tr>
<td>Select</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if schema owner, or if granted the corresponding privilege.</td>
</tr>
<tr>
<td>Create/Drop/Alter table</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if schema owner, or if granted the corresponding privilege.</td>
</tr>
<tr>
<td>Create/Drop schema</td>
<td>Yes</td>
<td>No</td>
<td>Yes for CREATE SCHEMA, if the schema name is the same as the user and does not yet exist.</td>
</tr>
<tr>
<td>Object</td>
<td>Splice User</td>
<td>Regular User</td>
<td>Role</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------</td>
<td>--------------</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>Has Permission?</td>
<td>Can Grant or Revoke?</td>
<td>Has Permission?</td>
</tr>
<tr>
<td>System Procedures/Routines</td>
<td>Yes</td>
<td>No</td>
<td>Yes for system schemas other than SYSCS_UTIL and SQLJ, which require that execution privilege be explicitly granted.</td>
</tr>
</tbody>
</table>
Using Apache Ranger with Splice Machine

Apache Ranger is a centralized security framework that allows you to manage fine-grained access control over Hadoop and related components. The Splice Machine Ranger plug-in extends Ranger security management to your Splice Machine database.

You can use Apache Ranger to:

- Manage policies for accessing resources by specific users and/or groups
- Audit tracking
- Analyze policies to gain deeper control of your system
- Delegate administration of certain data to other group owners

Ranger is currently only available for customers running the Enterprise version of Splice Machine on Hortonworks.

The remainder of this topic describes using Ranger with Splice Machine in these sections:

- Installing Ranger for Splice Machine
- Ranger Components
- Establishing Splice Machine Security Policies with Ranger
- Using Ranger with LDAP
- Using Ranger with Kerberos
- Reviewing Audit Logs

Installing Ranger for Splice Machine

You can install Apache Ranger with the Splice Machine Ambari Service on Hortonworks clusters that are running supported software versions, as listed below. The instructions for installing Ranger are included in the Splice Machine installation instructions in the docs subdirectory of the GitHub directory for each product/platform version:

<table>
<thead>
<tr>
<th>Splice Machine Version</th>
<th>Platform Version</th>
<th>Install Instructions URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>HDP 2.6.4</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.6.4/docs/HDP-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.6.4/docs/HDP-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>HDP 2.6.3</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.6.3/docs/HDP-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.6.3/docs/HDP-installation.md</a></td>
</tr>
</tbody>
</table>
Establishing Splice Machine Security Policies with Ranger

The instructions in this section assume that you already have:

- Used our instructions to install the Splice Machine Ranger plug-in
- Configured basic audit and security settings.
- Added the `splicemachine` service in Ranger on one of your Region Servers

You can now establish security policies for your database in two steps:

1. Create users and groups in your Splice Machine database
2. Use the Ranger Administrative user interface (via Ambari) to create policies that apply to those users and groups.

To access this user interface:
a. Select **Ranger** in Ambari.

b. In the main Ranger screen, select **Ranger Admin UI** under the **Quick Links** pull-down.

As indicated in our installation instructions, you **must** create a policy that allows your database users to **execute** routines in the **SYSIBM** schema: Splice Machine depends on execution of these routines for database operations. If you've not yet done so, follow the instructions in the next section.

**Setting Up the SYSIBM Policy**

If you've not already configured a Ranger policy that allows your Splice Machine database users to execute routines in the **SYSIBM** schema, follow these steps:

1. Access the Ranger Admin UI.

2. In the **Service Manager**, click the small, green **splicemachine** service link:

   ![Service Manager screenshot](image)

   This displays the list of policies defined for your **splicemachine** service. The initial list of policies were created by default for the **splice** administrative user.
3. Click the Add New Policy button:

4. Create a Schema policy named SYSIBM that allows users to execute all (*) of the routines in that schema. In this screenshot, you'll notice that, for demonstration purposes, we have only applied this policy to a user named BOB who is already defined in our database:
As you can see, each policy that you create in Ranger applies to specific object types (tables, UDTs, routines, sequences, etc.) in a specific schema. You can also create policies that apply to certain columns of a table. Each policy specifies which group or user the policy applies to, and which permissions (All, Select, Update, Insert, Trigger, Execute, etc.) the user(s) have for the specified entity.

Creating Additional Policies
To add new policies for your database users, you need to:

1. Add the user in your database, if you've not already done so. You can use the Splice Machine SYCS_UTIL.SYSCS_CREATE_USER system procedure to add a new user; for example:
   ```sql
   splice> CALL SYCS_UTIL.SYSCS_CREATE_USER('myUserId', 'MyPswd');
   Statement executed.
   ```

   **NOTE:** If you're using LDAP with Splice Machine, you don't need to create a user in your Splice Machine database; instead, you can simply make sure the user name in your Ranger configuration exactly matches the user name in your LDAP configuration. See the Using Ranger with LDAP section below for details.

2. Create a policy in the Ranger Admin UI, as shown in the Setting Up the SYSIBM Policy section, above. Specify the new user's name and the permission you want to grant them in the new policy. This screenshot shows an example of granting user BOB permission to select from TABLE_1 in the CDL schema in a Splice machine database:
Note that because this user has only been granted `select` permission on the table, he will not be allowed to perform other operations on this table, such as inserting or deleting.

3. Log into Splice Machine as the user:
   ```
sqlshell.sh -u myUserId -s MyPswd
Running Splice Machine SQL shell
splice>
   ```

**Using Ranger with LDAP**

When you use Ranger with LDAP, you don't need to create a user in your Splice Machine database; you just need to make sure that the user name in your Ranger configuration matches the LDAP user name.

**Beware:** LDAP is not case sensitive and converts user names to uppercase. Since Splice Machine is case sensitive, you must specify the Ranger user name in uppercase for it to correctly match the LDAP name in Splice Machine.

**Using Ranger with Kerberos**

There are some additional changes you need to make if you’re using Ranger in a Kerberized environment:

1. You must add the following three configuration properties for Splice Machine in the Ranger user interface:
   ```
   Add New Configurations
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag.download.auth.users</td>
<td>hbase</td>
</tr>
<tr>
<td>policy.download.auth.users</td>
<td>hbase</td>
</tr>
<tr>
<td>policy.grantrevoke.auth.users</td>
<td>hbase</td>
</tr>
</tbody>
</table>
   ```

2. You must specify a fully qualified domain name (e.g. `www.mydomain.com`) instead of an IP address in the following property in Ambari’s SpliceMachine service configuration:
   ```
   ranger.plugin.splicemachine.policy.rest.url
   ```

**Reviewing Audit Logs**

You can examine the logs in Ranger:
1. In Ranger, select the Audit tab.

2. Enter a start date and specify splicemachine as the service name.

3. View the log.

You can also examine the logs in HDFS. These log files are found in a subdirectory of /ranger/audit/splicemachine; for example, /ranger/audit/splicemachine/20180514.
Accessing Splice Machine from Windows on a Kerberized Cluster

This section shows you how to use either of these Kerberos implementations to access Splice Machine from a computer running *Microsoft Windows*:

- Configuring Microsoft Active Directory Kerberos
- Configuring MIT Kerberos

**Configuring Microsoft Active Directory Kerberos**

To use Microsoft Active Directory Kerberos with Splice Machine, you need to do the following:

1. Configure our ODBC driver to use Microsoft Active Directory Kerberos; you typically do this during driver installation.
2. Verify that MIT Kerberos *is not installed* on the client Windows computer.
3. Make sure that the MIT Kerberos Hadoop realm has been configured to trust the Active Directory realm so that users in the Active Directory realm can access services in the MIT Kerberos Hadoop realm.

**Configuring MIT Kerberos**

To use MIT Kerberos with the Splice Machine on Windows, you must download and install *MIT Kerberos for Windows 4.0.1*. Follow these steps:

1. Download and Run the MIT Kerberos Installer
2. Set up the Kerberos Configuration File
3. Set up the Kerberos Credential Cache File
4. Obtain a ticket for a Kerberos Principal

**Step 1: Download and Run the MIT Kerberos Installer for Windows**

You can find the installer here: [http://web.mit.edu/kerberos/dist/kfw/4.0/kfw-4.0.1-amd64.msi](http://web.mit.edu/kerberos/dist/kfw/4.0/kfw-4.0.1-amd64.msi).


**Step 2: Set up the Kerberos Configuration file**

There are two ways to do this, both of which are described in this section.
Set up in the default windows directory.

Set up in a custom location.

Set Up the Configuration in the Default Windows Directory
Follow these steps to set up your configuration file in the default directory:

1. Obtain the `krb5.conf` configuration file from your Kerberos administrator.
2. Rename that file to `krb5.ini`.
3. Copy the `krb5.ini` file to the `C:\ProgramData\MIT\Kerberos5` directory.

Set Up the Configuration in a Custom Location
Follow these steps to set up the configuration in a custom location:

1. Obtain the `/etc/krb5.conf` configuration file from your Kerberos administrator.
2. Place the `krb5.conf` file in an accessible directory and make note of the full path name.
3. Click Start, then right-click Computer, and then click Properties.
4. Click Advanced system settings.
5. In the System Properties dialog, click the Advanced tab, and then click Environment Variables.
6. In the Environment Variables dialog, under the System variables list, click New.
7. In the New System Variable dialog, in the Variable Name field, type `KR5_CONFIG`.
8. In the Variable Value field, type the absolute path to the `krb5.conf` file from step 1.
9. Click OK to save the new variable.
10. Ensure the variable is listed in the System variables list.
11. Click OK to close the Environment Variables dialog, and then click OK to close the System Properties dialog.

Step 3: Set Up the Kerberos Credential Cache File
Kerberos uses a credential cache to store and manage credentials. Follow these steps to set up the credentials cache file:
1. Create the directory where you want to save the Kerberos credential cache file; for example, you can use C:\temp.

2. Click Start, then right-click Computer, and then click Properties.

3. Click Advanced system settings.

4. In the System Properties dialog, click the Advanced tab, and then click Environment Variables.

5. In the Environment Variables dialog, under the System variables list, click New.

6. In the New System Variable dialog, in the Variable Name field, type KRB5CCNAME.

7. In the Variable Value field, type the path to the folder you created in step 1, and then append the file name krb5cache. For example, C:\temp\krb5cache.

   **NOTE:** krb5cache is a file (not a directory) that is managed by the Kerberos software which should not be created by users; if you receive a permission error when you first use Kerberos, ensure that krb5cache does not already exist as a file or directory.

8. Click OK to save the new variable.

9. Ensure the variable appears in the System variables list.

10. Click OK to close the Environment Variables dialog, and then click OK to close the System Properties dialog.

11. To ensure that Kerberos uses the new settings, **restart your computer.**

---

**Step 4: Obtain a Ticket for a Kerberos Principal**

A principal is a user or service that can authenticate to Kerberos. To authenticate to Kerberos, a principal must obtain a ticket in one of these ways:

- Obtain a ticket using a password.
- Obtain a ticket using the default keytab file.
- Obtain a ticket using a custom keytab file.

Each of these options is described in this section.
**Obtain a Ticket Using a Password**

1. Click the **Start** button, then click **All Programs**, and then click the **Kerberos for Windows (64-bit)** or the **Kerberos for Windows (32-bit)** program group.

2. Click **MIT Kerberos Ticket Manager**.

3. In the MIT Kerberos Ticket Manager, click **Get Ticket**.

4. In the Get Ticket dialog, type your principal name and password, and then click **OK**.

   If the authentication succeeds, then your ticket information appears in the MIT Kerberos Ticket Manager.

**Obtain a Ticket Using the Default Keytab File**

1. Click the **Start button > All Programs > Accessories > Command Prompt**

2. In the Command Prompt prompt, type a command using the following syntax:

   ```
   kinit -k principal
   ```

   **principal** is the Kerberos principal to use for authentication. For example:

   ```
   my/myserver.example.com@EXAMPLE.COM
   ```

   If the cache location **KRB5CCNAME** is not set or not used, then use the **-c** option of the `kinit` command to specify the credential cache. The **-c** argument must appear last on the command line. For example:

   ```
   kinit -k mydir/fully.qualified.domain.name@your-realm.com -c C:\Program Data\MIT\krbcache
   ```

**Obtain a Ticket Using a Custom Keytab File**

1. Click the **Start button > All Programs > Accessories > Command Prompt**.

2. In the Command Prompt, type a command using the following syntax:

   ```
   kinit -k -t *keytab_file* principal
   ```
keytab_file is the full path to the keytab file. For example:

```
C:\mykeytabs\myserver.keytab
```

principal is the Kerberos principal to use for authentication. For example:

```
mydir/myserver.example.com@EXAMPLE.COM
```

If the cache location KRB5CCNAME is not set or not used, then use the -c option of the kinit command to specify the credential cache. The -c argument must appear last on the command line. For example:

```
kinit -k -t C:\mykeytabs\myserver.keytab mydir/fully.qualified.domain.name@your-realm.com -c C:\ProgramData\MIT\krbcache
```

**NOTE:** For more information about configuring Kerberos, consult the MIT Kerberos documentation: [http://web.mit.edu/kerberos/](http://web.mit.edu/kerberos/).
Splice*Plus (PL/SQL)

This section describes Splice Machine’s implementation of the PL/SQL language and SplicePlus*, the PL/SQL interpreter for Splice Machine.

**ENTERPRISE ONLY:** This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product; contact Splice Machine Sales for information.

This section contains the following topics about using Splice*Plus, PL/SQL for Splice Machine:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Describes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Using Splice*Plus</strong></td>
<td>How to invoke and use Splice*Plus</td>
</tr>
<tr>
<td><strong>The Splice*Plus Language</strong></td>
<td>A summary of the Splice*Plus PL/SQL language implementation.</td>
</tr>
</tbody>
</table>
Using Splice*Plus

Splice*Plus extends Splice Machine SQL with a procedural language you can use to perform operations that you would otherwise write as SQL queries.

Splice*Plus is a command line utility program that provides an interpreter for running PL/SQL programs interactively. You can run PL/SQL programs stored in text files, or you can enter PL/SQL statements and blocks interactively, similarly to the way you use the splice> prompt to run SQL commands.

If you are already familiar with PL/SQL from using it with Oracle or another database, Splice Machine PL/SQL is almost exactly the same. The Splice*Plus language topic provides a quick review of the supported language structures.

The remainder of this topic includes these sections:

- Invoking Splice*Plus describes how you can invoke Splice*Plus on the command line
- The Splice*Plus Analyzer
- Hello World in PL/SQL presents a simple PL/SQL program
- Storing PL/SQL Programs in Your Database

Invoking Splice*Plus

Splice*Plus is automatically installed for you when you install the Enterprise version of Splice Machine. You can invoke Splice*Plus to run one or more PL/SQL programs, or you can use it interactively by invoking it without any program files on the command line. The basic syntax is:

```
spliceplus [--output outputFile] [cmdlineOptions*] sqlFile*
```

For example, this command line runs the PL/SQL program in the file myExample1.sql:

```
spliceplus myExample1.sql
```

You can add the --output option to send any program output to a file; for example:

```
spliceplus --output myExample1.out myExample1.sql
```

You can also invoke Splice*Plus to display help or version information from the command line:

```
spliceplus --help
spliceplus --version
```

Command Line Options

You can specify zero or more of the following options on the Splice*Plus command line, each of which have default values that are applied if the option is not specified on the command line. Here's the syntax:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>debugOptions</td>
<td>Specifies debugging options. You can specify any number of these options:</td>
<td>OFF</td>
</tr>
<tr>
<td></td>
<td>- SCRIPT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- LOCAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- GLOBAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- OPCODE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- FUNCALL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- FUNARGS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- FUNAST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- SQL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- SQLDATA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- SQLANALYZE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- SQLCURSOR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- JDBC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- JDBC.AUTH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- DISABLE_PERSISTENCE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- PERSISTENCE</td>
<td></td>
</tr>
<tr>
<td>jdbcString</td>
<td>The JDBC connection string to use, in this form:</td>
<td>jdbc://localhost:1527/splicedb</td>
</tr>
<tr>
<td></td>
<td>//hostname:1527/splicedb</td>
<td></td>
</tr>
<tr>
<td>schemaName</td>
<td>The name of the database schema to use.</td>
<td>SPLICE</td>
</tr>
</tbody>
</table>

**The Splice*Plus Analyzer**

You can use the `spliceplus-analyzer` program to profile which features your PL/SQL file uses; this allows you to analyze which features are supported in Splice*Plus.

The basic syntax for running the analyzer is:

```
spliceplus-analyzer [--output outputFile] [--verbose] sqlFile+
```

You can use the `--output` option to save the analyzer output to a CSV file. The `--verbose` option increases the amount of information the analyzer displays.

You can also invoke the analyzer to display help or version information from the command line:
Analyzer Example
You can analyze the newtroot.sql example program with this command line:

```
spliceplus-analyzer ./examples/newtroot.sql
```

You'll see output like this:

```
Analyzing ./examples/newtroot.sql
Feature count is:
PlConstruct:ArgumentPositional:Implemented x 3
PlConstruct:AssignmentStatement:Implemented
PlConstruct:BlockStatement:Implemented
PlConstruct:LoopWhile:Implemented
PlType:SimpleDouble:Implemented x 3
PlExpression:Add:Implemented
PlExpression:Constant:Implemented x 4
PlExpression:Div:Implemented x 2
PlExpression:Gt:Implemented x 2
PlExpression:Identifier:Implemented x 5
PlExpression:LogicalAnd:Implemented
PlExpression:Lt:Implemented
PlExpression:Mul:Implemented
PlExpression:PlsqlFunction:Implemented x 3
PlExpression:Sub:Implemented
PlBuiltinFunctionCall:ABS:PartiallyImplemented
PlBuiltinFunctionCall:DBMS_OUTPUT.PUT_LINE:PartiallyImplemented
PlUserFunctionCall:CW.ASSERT_TRUE:Implemented
```

Hello World in PL/SQL
Here's a basic PL/SQL version of "Hello World":

```
DECLARE
    message  varchar2(20):= 'Hello, World!';
BEGIN
    dbms_output.put_line(message);
EXCEPTION
    WHEN PROGRAM_ERROR THEN
        dbms_output.put_line('Uh-oh; something went wrong');
END;
/
```
NOTE: The single / at the end of the program block is required to tell the PL/SQL interpreter that it should run the code block; if you leave the / out, the interpreter will simply wait for more input after loading the block.

**Storing Your PL/SQL Code**

All PL/SQL procedures that you create with Splice*Plus are stored in the SYS.SYSSOURCECODE table in your database.

You can specify that your code should *not* be stored in your database with the `--debug DISABLE_PERSISTENCE` command line option.
Splice*Plus - PL/SQL for Splice Machine

This topic describes Splice Machine's implementation of the PL/SQL language and SplicePlus*, the PL/SQL interpreter for Splice Machine.

**ENTERPRISE ONLY:** This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product; contact Splice Machine Sales for information.

About PL/SQL

This section provides only the briefest summary of the PL/SQL language; instead, we recommend that you check the Oracle documentation site for their documentation on PL/SQL, or use one of the many third party books that are available. One excellent starting point, if you're new to PL/SQL, is this online tutorial: [www.tutorialspoint.com/plsql](http://www.tutorialspoint.com/plsql).

PL/SQL Basics

PL/SQL is a procedural programming language that is tightly integrated with SQL. Which means that you can use PL/SQL to create programs that perform SQL operations. The syntax of PL/SQL is based on the programming languages ADA and Pascal.

The basic unit of a PL/SQL operation is the block, which groups together related declarations and statements. Blocks can be procedures, functions, or anonymous blocks, which are typically used for the main body of your programs. The Hello World example we showed above is an example of an anonymous block.

You define a PL/SQL block with a declaration section, a procedural section, and an exception section. The following is an example of an anonymous block:

```sql
<<Main>>
DECLARE
    myNum number(2);
    myText varchar2(12) := 'Hello world';
    myDate date := SYSDATE;
BEGIN
    SELECT name
        INTO myText
        FROM playerNames
        WHERE ID = 27;
EXCEPTION
    WHEN others THEN
        dbms_output.put_line('Error Message is ' || sqlerrm);
END;/
```

Some notes of interest about the basic constructs of PL/SQL:
<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Labels</strong></td>
<td>Labels are specified surrounded by double chevrons (&lt;&lt; and &gt;&gt;). Labels are optional for anonymous blocks such as the main body of your program.</td>
</tr>
<tr>
<td><strong>Declarations</strong></td>
<td>The declare section of a block is optional; however, you must declare a variable prior to using it.</td>
</tr>
<tr>
<td></td>
<td>The declaration section of an anonymous block begins with the keyword DECLARE.</td>
</tr>
<tr>
<td></td>
<td>Variable declarations can include initial values, which you assign with the PL/SQL assignment operator (:=). If you don't specify an initial value, the variable is initially NULL.</td>
</tr>
<tr>
<td><strong>Terminators</strong></td>
<td>You must terminate each declaration and statement with a semicolon (;).</td>
</tr>
<tr>
<td><strong>Executable part</strong></td>
<td>The executable part of the block begins with the BEGIN keyword and is required.</td>
</tr>
<tr>
<td><strong>Exceptions</strong></td>
<td>The exception keyword begins the optional section of the block in which you handle exceptions generated in the executable part.</td>
</tr>
<tr>
<td><strong>Nested blocks and variable scope</strong></td>
<td>You can nest a block within another block.</td>
</tr>
<tr>
<td></td>
<td>Variable scoping works as you expect: variables declared in outer blocks are accessible to all blocks nested within them (inner blocks), while variables declared in inner blocks are not accessible to the outer blocks.</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>You can add comments in two ways:</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt; single-line comments start with the delimiter -- (double hyphen)</td>
</tr>
<tr>
<td></td>
<td>&gt;&gt; multi-line comments are enclosed by /* and */.</td>
</tr>
<tr>
<td><strong>/ at end</strong></td>
<td>You include a single / on a blank line at the end of a PL/SQL block to tell the PL/SQL interpreter to run the block after loading it. If you leave this out, the interpreter will simply wait after loading your code block.</td>
</tr>
</tbody>
</table>

**NOTE:** If you're creating a program that includes one or more function or procedure definitions (see below), you must include the / after each function or procedure block to have them loaded when you run your program.
PL/SQL Procedures and Functions

Your PL/SQL programs can include procedures and functions, which are types of blocks. Procedures do not return values; each function returns a single value.

Like anonymous blocks, functions and procedures include:

- An optional declaration section, which does not begin with the DECLARE keyword
- An executable section
- An optional exception section

The syntax for specifying functions and procedures is:

```sql
CREATE [OR REPLACE] FUNCTION function_name
   ([parameter_name [IN | OUT | IN OUT] type [, ...]])
RETURN return_datatype
   {IS | AS}
BEGIN
   < function_body >
END [function_name];

CREATE [OR REPLACE] PROCEDURE procedure_name
   ([parameter_name [IN | OUT | IN OUT] type [, ...]])
   {IS | AS}
BEGIN
   < procedure_body >
END [procedure_name];
```

As mentioned, you can define your local variables in a procedure or function just as you do for anonymous blocks; however, the DECLARE keyword is not used. For example:

```sql
CREATE OR REPLACE FUNCTION GetDisplayString( prefixStr IN VARCHAR(50), numVal IN number) RETURN varchar2(60) IS
defaultPrefix varchar2(30) := "The number is: ";
BEGIN
   IF (prefixStr = '') THEN prefixStr := defaultPrefix; END IF;
RETURN prefixStr || numVal;
END;/
```

Nested Functions and Procedures

You can nest blocks, including function and procedure blocks inside other blocks. Standard scoping rules for variables and named blocks: any nested (inner) block has access to the definitions available in its containing block, but outer blocks cannot access definitions in inner blocks.

For example, here's an anonymous block that declares a function definition and then uses that function:
DECLARE
  prefix VARCHAR(50);
FUNCTION GetDisplayString( prefixStr IN VARCHAR(50), numVal IN NUMBER)
  RETURN VARCHAR2(60)
IS
  defaultPrefix VARCHAR2(30) := "The number is: ";
  negNumException EXCEPTION;
BEGIN
  IF (prefixStr = '') THEN
    prefixStr := defaultPrefix;
  END IF;
  IF (numVal < 0) THEN
    RAISE 'negativeNum'
  RETURN( prefixStr || numVal );
END;
BEGIN
  prefix := "The number you want is: ";
  dbms_output.put_line(GetDisplayString(prefix, 13));
END;
END;/

Handling Exceptions in PL/SQL

PL/SQL makes exception handling easy. Each block can optionally contain an exception handling block that can contain any number of WHEN statements. In summary:

- You can raise exceptions by name in your code:
  
  RAISE <exception_name>;

- Each block can contain an optional exception section that begins with the exception keyword.

- The exception section can contain any number of exception handlers, each of which is defined in a WHEN statement.

- Each WHEN statement handles a specific exception_name:
  
  WHEN <exception_name>;

- The exception name others is used to define an exception handler that handles any exceptions not specifically handled by another WHEN statement.
See Also
To learn more about PL/SQL, consider these resources:

- The Oracle documentation site.
- Third party PL/SQL books available at your favorite bookstore or online.
# Tuning and Debugging Query Performance

This section contains the following topics to help you optimize your Splice Machine database queries:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Describes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizing Queries</td>
<td>Gets you started with optimizing your queries for Splice Machine.</td>
</tr>
<tr>
<td>Using Statistics</td>
<td>Using our statistics gathering facility to help optimize queries.</td>
</tr>
<tr>
<td>Explain Plan</td>
<td>The Explain Plan feature, which allows you to examine the execution plan for a query without executing the query.</td>
</tr>
<tr>
<td>Logging</td>
<td>The Splice Machine logging facility.</td>
</tr>
<tr>
<td>Debugging</td>
<td>Describes the parameter values to use when using debugger software with Splice Machine.</td>
</tr>
<tr>
<td>Using Snapshots</td>
<td>How to use the Splice Machine snapshot feature to create a restorable snapshot of a table or schema</td>
</tr>
</tbody>
</table>

For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](https://github.com/SpliceMachine).
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.
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*:

```sql
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::

```sql
splice> explain select * from sys.systables where id=1;
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
```
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,parti
tions=1)
    ->  ProjectRestrict(n=2,totalCost=4.054,outputRows=10,outputHeapSize=240 B,parti
tions=1,preds=[like(TABLENAME[0:1], %SYS%)])
    ->  IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=4.054,outputRows=20,outputH
eapSize=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 K
B,partitions=1)
    ->  IndexLookup(n=2,totalCost=78.715,outputRows=18,outputHeapSize=921.586 KB,par
titions=1)
    ->  IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=6.715,outputRows=18,outputH
eapSize=921.586 KB,partitions=1,baseTable=SYSTABLES(48),preds=[(TABLENAME[1:2] = SYS
TABLES)])

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,p
partitions=1)
        ->  BroadcastJoin(n=3,totalCost=12.648,outputRows=20,outputHeapSize=6.641 KB,par
        ->  TableScan[SYSSCHEMAS(32)](n=2,totalCost=4.054,outputRows=20,outputHeapSiz
e=6.641 KB,partitions=1)
        ->  TableScan[SYSTABLES(48)](n=1,totalCost=4.054,outputRows=20,outputHeapSize
e=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.systablest 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, partitions=1)
    ->  ProjectRestrict(n=4, totalCost=130.579, outputRows=20, outputHeapSize=2 MB, partitions=1)
    ->  BroadcastLeftOuterJoin(n=3, totalCost=130.579, outputRows=20, outputHeapSize=2 MB, partitions=1, preds=[(S.SCHEMAID[4:1] = T.SCHEMAID[4:4])])
      ->  IndexScan[SYSTABLES_INDEX1(145)](n=2, totalCost=7.017, outputRows=20, outputHeapSize=2 MB, partitions=1, baseTable=SYSTABLES(48))
      ->  TableScan[SYSSCHEMAS(32)](n=1, totalCost=7.516, outputRows=20, outputHeapSize=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 from 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, partitions=1)
    ->  Union(n=3, totalCost=12.356, outputRows=40, outputHeapSize=1.094 KB, partitions=1)
      ->  IndexScan[SYSSCHEMAS_INDEX1(209)](n=2, totalCost=4.054, outputRows=20, outputHeapSize=1.094 KB, partitions=1, baseTable=SYSSCHEMAS(32))
      ->  IndexScan[SYSTABLES_INDEX1(145)](n=1, totalCost=4.054, outputRows=20, outputHeapSize=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,partition=1)
      --> OrderBy(n=2,totalCost=12.356,outputRows=20,outputHeapSize=480 B,partition=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,partition=16)
      --> ProjectRestrict(n=4,totalCost=8.32,outputRows=20,outputHeapSize=480 B,partition=16)
         --> GroupBy(n=3,totalCost=4.054,outputRows=20,outputHeapSize=480 B,partition=1)
            --> ProjectRestrict(n=2,totalCost=4.054,outputRows=20,outputHeapSize=480 B,partition=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,partitions=1)
        -> ProjectRestrict(n=2,totalCost=4.054,outputRows=20,outputHeapSize=3.32 KB,partitions=1)
          -> IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=4.054,outputRows=20,outputHeapSize=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,partitions=1)
    -> ProjectRestrict(n=4,totalCost=4.054,outputRows=20,outputHeapSize=480 B,partitions=1)
      -> Subquery(n=3,totalCost=12.55,outputRows=20,outputHeapSize=480 B,partitions=1,correlated=true,expression=true,invariant=true)
        -> IndexScan[SYSTABLES_INDEX1(145)](n=2,totalCost=4.054,outputRows=20,outputHeapSize=480 B,partitions=1,baseTable=SYSTABLES(48),preds=[(T2.TABLENAME[0:1] = T.TABLENAME[4:1])])
          -> IndexScan[SYSTABLES_INDEX1(145)](n=1,totalCost=4.054,outputRows=20,outputHeapSize=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. This topic contains these sections:

- **Using Logging**
- **Configure Individual Logger Objects to Log**
- **SQL Logger Functions**

**Using Logging**

Splice Machine uses the open source [Apache log4j Logging API](https://logging.apache.org/log4j/2.x/), 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.

The remainder of this section shows you how to:

- **Mask Sensitive Information in Log Messages**
- **Manually Disable Logging**
- **Set Logger Levels**

**Masking Sensitive Information**

You can configure `log4j` to prevent sensitive information such as passwords and credit card information from being logged in log messages. To do so, you:

- Use the `com.splicemachine.utils.logging.MaskPatternLayout log4j layout pattern.`
- Specify a regular expression in `MaskPattern` that matches the part of log messages you want matched. * When logging with this layout, log4j will replace any text that matches the filter with this text: `MASKED SENSITIVE INFO`

For example:

```properties
log4j.appenders.splice Derby.layout=com.splicemachine.utils.logging.MaskPatternLayout
log4j.appenders.splice Derby.layout.ConversionPattern=%d{ISO8601} Thread[%t] %m%n
```
Given that layout, the following statement:

```
splice> INSERT INTO a VALUES myName, myPassword
```

will be logged as:

```
INSERT INTO a VALUES MASKED SENSITIVE INFO, MASKED SENSITIVE INFO
```

### 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:

- 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.

The following table displays the logger levels from lowest level to the highest:

<table>
<thead>
<tr>
<th>Logger Level</th>
<th>What gets logged for a logger object set to this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE</td>
<td>Captures all messages.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>Captures any message whose level is DEBUG, INFO, WARN, ERROR, or FATAL.</td>
</tr>
</tbody>
</table>
### Logger Level

<table>
<thead>
<tr>
<th>Logger Level</th>
<th>What gets logged for a logger object set to this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO</td>
<td>Captures any message whose level is INFO, WARN, ERROR, or FATAL.</td>
</tr>
<tr>
<td>WARN</td>
<td>Captures any message whose level is WARN, ERROR, or FATAL.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Captures any message whose level is ERROR or FATAL.</td>
</tr>
<tr>
<td>FATAL</td>
<td>Captures only messages whose level is FATAL.</td>
</tr>
</tbody>
</table>

### 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:

<table>
<thead>
<tr>
<th>Logger Name</th>
<th>Default Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.apache</td>
<td>ERROR</td>
<td>Logs all Apache software messages</td>
</tr>
<tr>
<td>com.splicemachine.db</td>
<td>WARN</td>
<td>Logs all Derby software messages</td>
</tr>
<tr>
<td>com.splicemachine.db.shared.common.sanity</td>
<td>ERROR</td>
<td>Logs all Derby Sanity Manager messages</td>
</tr>
<tr>
<td>com.splicemachine.derby.impl.sql.catalog</td>
<td>WARN</td>
<td>Logs Derby SQL catalog/dictionary messages</td>
</tr>
<tr>
<td>com.splicemachine.db.impl.sql.execute.operations</td>
<td>WARN</td>
<td>Logs Derby SQL operation messages</td>
</tr>
<tr>
<td>org.apache.zookeeper.server.ZooKeeperServer</td>
<td>INFO</td>
<td>Used to determine when Zookeeper is started</td>
</tr>
<tr>
<td>org.apache.zookeeper.server.persistence.FileTxnSnapLog</td>
<td>INFO</td>
<td>Logs Zookeeper transactions</td>
</tr>
<tr>
<td>com.splicemachine</td>
<td>WARN</td>
<td>By default, controls all Splice Machine logging</td>
</tr>
<tr>
<td>Logger Name</td>
<td>Default Level</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>com.splicemachine.derby.hbase.SpliceDriver</td>
<td>INFO</td>
<td>Prints start-up and shutdown messages to the log and to the console.</td>
</tr>
<tr>
<td>com.splicemachine.derby.management.StatementManager</td>
<td>ERROR</td>
<td>Set the level of this logger to TRACE to record execution time for SQL statements</td>
</tr>
</tbody>
</table>

To see a full list of loggers, use the `SYCS_UTIL.SYCS_GET_LOGGERS` system procedure:

```
CALL SYCS_UTIL.SYCS_GET_LOGGERS();
```

**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:

<table>
<thead>
<tr>
<th>System Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYCS_UTIL.SYCS_GET_LOGGERS</td>
<td>Displays a list of the active loggers.</td>
</tr>
<tr>
<td>SYCS_UTIL.SYCS_GET_LOGGER_LEVEL</td>
<td>Displays the current logging level of the specified logger.</td>
</tr>
<tr>
<td>SYCS_UTIL.SYCS_SET_LOGGER_LEVEL</td>
<td>Sets the current logging level of the specified logger.</td>
</tr>
</tbody>
</table>

**See Also**

- `SYCS_UTIL.SYCS_GET_LOGGERS`
- `SYCS_UTIL.SYCS_GET_LOGGER_LEVEL`
- `SYCS_UTIL.SYCS_SET_LOGGER_LEVEL`
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:

```plaintext
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
---|---
$n=number$ | The number of nodes.
$rows=number$ | The number of output rows.
$updateMode=[mode]$ | The update mode of the statement.
$engine=[engineType]$ | engineType=Spark means this query will be executed by our OLAP engine.
| engineType=control means this query will be executed by our OLTP engine.

Each node row in the output contains the following fields:

Field | Description
---|---
$[node label]$ | The name of the node
$n=number$ | The result set number.
| This is primarily used internally, and can also be used to determine the relative ordering of optimization.
$totalCost=number$ | The total cost to perform this operation.
| This is computed as if the operation is at the top of the operation tree.
$processingCost=number$ | The cost to process all data in this node.
| Processing includes everything except for reading the final results over the network.
$transferCost=number$ | The cost to send the final results over the network to the control node.
$outputRows=number$ | The total number of rows that are output.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputHeapSize=number unit</td>
<td>The total size of the output result set, and the unit in which that size is expressed, which is one of:</td>
</tr>
<tr>
<td></td>
<td>&gt; B</td>
</tr>
<tr>
<td></td>
<td>&gt; KB</td>
</tr>
<tr>
<td></td>
<td>&gt; MB</td>
</tr>
<tr>
<td></td>
<td>&gt; GB</td>
</tr>
<tr>
<td></td>
<td>&gt; TB</td>
</tr>
<tr>
<td>partitions==number</td>
<td>The number of partitions involved.</td>
</tr>
<tr>
<td></td>
<td><strong>Partition</strong> is currently equivalent to <strong>Region</strong>; however, this will not necessarily remain true in future releases.</td>
</tr>
</tbody>
</table>

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) |
| | | --> 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**
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:

<table>
<thead>
<tr>
<th>Optimization Mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Explain Plan</td>
<td>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.</td>
</tr>
<tr>
<td>Use Statistics</td>
<td>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.</td>
</tr>
<tr>
<td>Use WHERE clauses</td>
<td>Use a WHERE clause in your queries to restrict how much data is scanned.</td>
</tr>
<tr>
<td>Use indexes</td>
<td>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.</td>
</tr>
<tr>
<td>Use Splice Machine query hints</td>
<td>The Splice Machine query optimizer allows you to provide hints to help in the optimization process.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Hint Type</th>
<th>Example</th>
<th>Used to Specify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td><code>--splice-properties index=my_index</code></td>
<td>Which index to use or not use</td>
</tr>
<tr>
<td>Join Order</td>
<td><code>--splice-properties joinOrder=fixed</code></td>
<td>Which join order to use for two tables</td>
</tr>
<tr>
<td>Join Strategy</td>
<td><code>--splice-properties joinStrategy=sortmerge</code></td>
<td>How a join is processed (in conjunction with the Join Order hint)</td>
</tr>
<tr>
<td>Spark</td>
<td><code>--splice-properties useSpark=true</code></td>
<td>That you want a query to run (or not run) on Spark</td>
</tr>
<tr>
<td>Delete</td>
<td><code>--splice-properties bulkDeleteDirectory='path'</code></td>
<td>That you are deleting a large amount of data and want to bypass the normal write pipeline to speed up the deletion.</td>
</tr>
</tbody>
</table>

### 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.
Hints can be used in two locations: after a table identifier or after a `FROM` clause. Some hint types can be use after a table identifier, and some can be used after a `FROM` clause:

<table>
<thead>
<tr>
<th>Hint after a:</th>
<th>Hint types</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table identifier</td>
<td>index</td>
<td><code>SELECT * FROM member_info m, rewards r, points p --SPLICE-PROPERTIES index=ie_point WHERE...</code></td>
</tr>
<tr>
<td></td>
<td>joinStrategy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>useSpark</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bulkDeleteDirectory</td>
<td></td>
</tr>
<tr>
<td>A FROM clause</td>
<td>joinOrder</td>
<td><code>SELECT * FROM --SPLICE-PROPERTIES joinOrder=fixed mytable1 e, mytable2 t WHERE e.id = t.parent_id;</code></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Hint</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>joinOrder=FIXED</td>
<td>splice&gt; SELECT * FROM --SPLICE-PROPERTIES joinOrder=fixed&gt; mytable1 e, mytable2 t&gt; WHERE e.id = t.parent_id;</td>
</tr>
<tr>
<td>joinOrder=UNFIXED</td>
<td>splice&gt; SELECT * from --SPLICE-PROPERTIES joinOrder=unfixed&gt; mytable1 e, mytable2 t WHERE e.id = t.parent_id;</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>JoinStrategy Value</th>
<th>Strategy Description</th>
</tr>
</thead>
</table>
| BROADCAST          | Read the results of the Right Result Set (RHS) into memory, then for each row in the left result set (LHS), perform a local lookup to determine the right side of the join. BROADCAST will only work if at least one of the following is true:  
  - There is at least one equijoin (=) predicate that does not include a function call.  
  - There is at least one inequality join predicate, the RHS is a base table, and the join is evaluated in Spark. |
### 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](https://www.splicemachine.com/docs/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 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 `SYCS_UTIL.SNAPSHOT_TABLE` system procedure to create a named snapshot for a table.
- Use the `SYCS_UTIL.SNAPSHOT_SCHEMA` system procedure to create a named snapshot for a schema.
- Use the `SYCS_UTIL.RESTORE_SNAPSHOT` system procedure to restore a table or schema from a named snapshot.
- Use the `SYCS_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.
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 `SYCS_UTIL.DISABLE_COLUMN_STATISTICS` built-in system procedure:

```sql
SYCS_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](https://www.splicemachine.com/docs/latest/javadoc/com/splicemachine/db/util/ExplainPlan.html) 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 `SYCS_UTIL.DROP_SCHEMA_STATISTICS` procedure to drop statistics for an entire schema. For example:

```
splice> CALL SYCS_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 `SYCS_UTIL.ENABLE_COLUMN_STATISTICS` procedure. For example:

```
CALL SYCS_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 `SYCS_UTIL.DISABLE_COLUMN_STATISTICS` procedure:

```sql
splice> CALL SYCS_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 View to view statistics for specific tables
- Query the `SYSCOLUMNSTATISTICS` System View to view statistics for specific table columns

See Also

- Splice Machine Data Assignments and Comparisons
- `SYCS_UTIL.COLLECT_SCHEMA_STATISTICS`
- `SYCS_UTIL.DISABLE_COLUMN_STATISTICS`
» SYSCS_UTIL.DROP_SCHEMA_STATISTICS
» SYSCS_UTIL.ENABLE_COLUMN_STATISTICS
» SYSCOLUMNSTATISTICS System View
» SYSTABLESTATISTICS System View
Ingesting and Streaming Data With Splice Machine

This section provides tutorials to help you stream data with Splice Machine, in these topics:

- Creating a Kafka Producer
- Configuring a Kafka Feed
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.
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 remainder of this topic, 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:

<table>
<thead>
<tr>
<th>Product</th>
<th>DB Console Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database-as-Service</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accumulators</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Additional Metrics</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>DAG Visualization</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Enable Zooming</strong></td>
<td>For event timeline 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.</td>
</tr>
<tr>
<td><strong>Event Timeline</strong></td>
<td>A view that graphically displays the sequence of all jobs, a specific job, or a stage within a job.</td>
</tr>
<tr>
<td><strong>Executor</strong></td>
<td>A process that runs tasks on a cluster node.</td>
</tr>
<tr>
<td><strong>GC Time</strong></td>
<td>The amount of time spent performing garbage collection in a stage.</td>
</tr>
<tr>
<td><strong>Job</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Locality Level</strong></td>
<td>To minimize data transfers, Spark tries to execute as close to the data as possible. The Locality Level value indicates whether a task was able to run on the local node.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Scheduling Mode</strong></td>
<td>The scheduling mode used for a job.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Scheduling Pool</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>ScrollInsensitive row</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Shuffling</strong></td>
<td>Shuffling is the reallocation of data between multiple stages in a Spark job.</td>
</tr>
<tr>
<td></td>
<td><em>Shuffle Write</em> is amount of data that is serialized and written at the end of a stage for transmission to the next stage. <em>Shuffle Read</em> is the amount of serialized data that is read at the beginning of a stage.</td>
</tr>
<tr>
<td><strong>Stage</strong></td>
<td>The Splice Machine Spark scheduler splits the execution of a job into stages, based on the RDD transformations required to complete the job.</td>
</tr>
<tr>
<td></td>
<td>Each stage contains a group of tasks that perform a computation in parallel.</td>
</tr>
<tr>
<td><strong>Task</strong></td>
<td>A computational command sent from the application driver to an executor as part of a stage.</td>
</tr>
</tbody>
</table>

**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*:

```
<table>
<thead>
<tr>
<th>Job Id (Job Group)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>113 (SPLICE &lt;387&gt;)</td>
<td>-- EXPLAIN QUERY 22 explain select cntrycode, count(*) as numcust, sum(c_acctbal) as totalrevenue as totalrevenue from sales where c_emailaddress is not null</td>
</tr>
<tr>
<td>Explain</td>
<td></td>
</tr>
</tbody>
</table>
```

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:

![Tab Bar Screenshot]

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:
Similarly, you can hover over the ? to display the definition for a term, like the definition of a job:

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.
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.
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Home</td>
<td>/Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Home/jre</td>
</tr>
<tr>
<td>Java Version</td>
<td>1.7.0_71 (Oracle Corporation)</td>
</tr>
<tr>
<td>Scala Version</td>
<td>version 2.10.4</td>
</tr>
</tbody>
</table>

**Spark Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver.source.splice-machine.class</td>
<td>com.splicemachine.derby.stream.spark.SpliceMachineSource</td>
</tr>
<tr>
<td>executor.source.splice-machine.class</td>
<td>com.splicemachine.derby.stream.spark.SpliceMachineSource</td>
</tr>
<tr>
<td>spark.app.id</td>
<td>application_1454971247434_0001</td>
</tr>
<tr>
<td>spark.app.name</td>
<td>SpliceMachine</td>
</tr>
<tr>
<td>spark.driver.appUIAddress</td>
<td><a href="http://192.168.1.15:4040">http://192.168.1.15:4040</a></td>
</tr>
<tr>
<td>spark.driver.cores</td>
<td>1</td>
</tr>
<tr>
<td>spark.driver.extraClassPath</td>
<td>/Users/garyhillerson/git/spliceengine/cdh5.4.1/splice_machine_test/ha llerson/git/spliceengine/cdh5.4.1/splice_machine_test/target/dep</td>
</tr>
<tr>
<td>spark.driver.host</td>
<td>localhost</td>
</tr>
<tr>
<td>spark.driver.maxResultSize</td>
<td>1g</td>
</tr>
<tr>
<td>spark.driver.memory</td>
<td>1g</td>
</tr>
<tr>
<td>spark.driver.port</td>
<td>58937</td>
</tr>
<tr>
<td>spark.enabled</td>
<td>true</td>
</tr>
<tr>
<td>spark.executor.cores</td>
<td>4</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Executor ID</th>
<th>Address</th>
<th>RDD Blocks</th>
<th>Storage Memory</th>
<th>Disk Used</th>
<th>Active Tasks</th>
<th>Failed Tasks</th>
<th>Complete Tasks</th>
<th>Total Tasks</th>
<th>Task Time</th>
<th>Input</th>
<th>Shuffle Read</th>
<th>Shuffle Write</th>
<th>Logs</th>
<th>Thread Dump</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>localhost:50141</td>
<td>0</td>
<td>0.0 B / 176.7 MB</td>
<td>0.0 B</td>
<td>0</td>
<td>8</td>
<td>30</td>
<td>38</td>
<td>11.0 s</td>
<td>709.0 B</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td>stdout stderr</td>
<td>Thread Dump</td>
</tr>
<tr>
<td>driver</td>
<td>192.168.1.15:50132</td>
<td>0</td>
<td>0.0 B / 736.1 MB</td>
<td>0.0 B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 ms</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td>stdout stderr</td>
<td>Thread Dump</td>
</tr>
</tbody>
</table>

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 | Share
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>163</td>
<td>query</td>
<td>-- QUERY 02 select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_c... Produce Result Set</td>
<td>2016/02/08 15:37:23</td>
<td>32 ms</td>
<td>1/1</td>
<td></td>
<td></td>
<td>18.3 KB</td>
</tr>
<tr>
<td>162</td>
<td>query</td>
<td>-- QUERY 02 select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_c... Row Limit: Fetch Limit 100</td>
<td>2016/02/08 15:37:23</td>
<td>45 ms</td>
<td>1/1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>query</td>
<td>-- QUERY 02 select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_c... Row Limit: Fetch Limit 100</td>
<td>2016/02/08 15:37:23</td>
<td>0.1 s</td>
<td>1/1</td>
<td></td>
<td></td>
<td>105 KB</td>
</tr>
<tr>
<td>160</td>
<td>query</td>
<td>-- QUERY 02 select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_c... Sort: Prepare Keys</td>
<td>2016/02/08 15:37:23</td>
<td>1 s</td>
<td>1/1</td>
<td></td>
<td></td>
<td>6.3 MB</td>
</tr>
<tr>
<td>158</td>
<td>query</td>
<td>-- 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</td>
<td>2016/02/08 15:37:22</td>
<td>1 s</td>
<td>1/1</td>
<td></td>
<td></td>
<td>6.2 MB</td>
</tr>
<tr>
<td>159</td>
<td>query</td>
<td>-- 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</td>
<td>2016/02/08 15:37:20</td>
<td>13 s</td>
<td>1/1</td>
<td></td>
<td></td>
<td>20.3 MB</td>
</tr>
<tr>
<td>157</td>
<td>query</td>
<td>-- QUERY 02 select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_c... Grouped Aggregate: Prepare Keys</td>
<td>2016/02/08 15:37:05</td>
<td>16 s</td>
<td>1/1</td>
<td></td>
<td></td>
<td>52.3 MB</td>
</tr>
</tbody>
</table>

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:
**Job Details Graphical Visualization View**

The DAG Visualization view for a job looks like this:
Details for Job 188

Status: RUNNING
Job Group: SPLICE <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:
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

<table>
<thead>
<tr>
<th>Metric</th>
<th>Min</th>
<th>25th percentile</th>
<th>Median</th>
<th>75th percentile</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>61 ms</td>
<td>0.1 s</td>
<td>0.1 s</td>
<td>0.2 s</td>
<td>0.3 s</td>
</tr>
<tr>
<td>Scheduler Delay</td>
<td>8 ms</td>
<td>9 ms</td>
<td>9 ms</td>
<td>10 ms</td>
<td>12 ms</td>
</tr>
<tr>
<td>Task Deserialization Time</td>
<td>1 ms</td>
<td>2 ms</td>
<td>2 ms</td>
<td>9 ms</td>
<td>10 ms</td>
</tr>
<tr>
<td>GC Time</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
</tr>
<tr>
<td>Result Serialization Time</td>
<td>0 ms</td>
<td>1 ms</td>
<td>1 ms</td>
<td>1 ms</td>
<td>1 ms</td>
</tr>
<tr>
<td>Getting Result Time</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
</tr>
<tr>
<td>Peak Execution Memory</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td>0.0 B</td>
<td>0.0 B</td>
</tr>
<tr>
<td>Input Size / Records</td>
<td>0.0 B / 0</td>
<td>0.0 B / 0</td>
<td>0.0 B / 0</td>
<td>3.9 KB / 0</td>
<td></td>
</tr>
</tbody>
</table>

Aggregated Metrics by Executor

<table>
<thead>
<tr>
<th>Executor ID</th>
<th>Address</th>
<th>Task Time</th>
<th>Total Tasks</th>
<th>Failed Tasks</th>
<th>Succeeded Tasks</th>
<th>Input Size / Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>localhost:51853</td>
<td>3 s</td>
<td>16</td>
<td>0</td>
<td>16</td>
<td>10.0 KB / 0</td>
</tr>
</tbody>
</table>

Tasks

The Event Time Line View of a Stage

The Event Timeline view of a stage looks like this:
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:

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 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 topic summarizes the hardware and software requirements for Splice Machine running on a cluster or on a standalone computer, in these sections:

- Cluster Node Requirements summarizes requirements for running Splice Machine on a cluster, including these subsections:
  - Amazon Web Services (AWS) Requirements
  - Hadoop Ecosystem Requirements
  - Linux Configuration Requirements
- Standalone Version Prerequisites summarizes requirements for running the standalone version of Splice Machine.
- Java JDK Requirements summarizes which Java JDK requirements for running Splice Machine.

This is an On-Premise-Only topic!

Cluster Node Requirements

The following table summarizes the minimum requirements for the nodes in your cluster:

<table>
<thead>
<tr>
<th>Component</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cores</strong></td>
<td>Splice Machine recommends that each node in your cluster have 8-12 hyper-threaded cores (16-32 hyper-threads) for optimum throughput and concurrency.</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>We recommend that each machine in your cluster have at least 64 GB of available memory.</td>
</tr>
<tr>
<td><strong>Disk Space</strong></td>
<td>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 ext4 file system.</td>
</tr>
<tr>
<td><strong>Hadoop Ecosystem</strong></td>
<td>The table in the next section, Hadoop Ecosystem Requirements, summarizes the specific Hadoop component versions that we support in each of our product releases.</td>
</tr>
</tbody>
</table>
### Software Tools and System Settings

The [Linux Configuration](#LinuxConf) section below summarizes the software tools and system settings required for your cluster machines.

<table>
<thead>
<tr>
<th>Component</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software Tools and System Settings</strong></td>
<td></td>
</tr>
<tr>
<td>Minimum Cluster Size</td>
<td>The minimum cluster size on AWS is 5 nodes:</td>
</tr>
<tr>
<td></td>
<td>• 1 master node</td>
</tr>
<tr>
<td></td>
<td>• 4 worker nodes</td>
</tr>
<tr>
<td>Minimum Node Size</td>
<td>Minimum recommended size of each node is <em>m4.4xlarge</em>.</td>
</tr>
<tr>
<td>Disk Space</td>
<td>Minimum recommended storage space:</td>
</tr>
<tr>
<td></td>
<td>• 100GB EBS root drive</td>
</tr>
<tr>
<td></td>
<td>• 4 EBS data drives per node</td>
</tr>
</tbody>
</table>
|                             | Note that the required number of data drives per node depends on your use case.

### Amazon Web Services (AWS) Requirements

If you're running on AWS, your cluster must meet these minimum requirements:

<table>
<thead>
<tr>
<th>Component</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Cluster Size</td>
<td>The minimum cluster size on AWS is 5 nodes:</td>
</tr>
<tr>
<td>Minimum Node Size</td>
<td>Minimum recommended size of each node is <em>m4.4xlarge</em>.</td>
</tr>
<tr>
<td>Disk Space</td>
<td>Minimum recommended storage space:</td>
</tr>
<tr>
<td></td>
<td>• 100GB EBS root drive</td>
</tr>
<tr>
<td></td>
<td>• 4 EBS data drives per node</td>
</tr>
</tbody>
</table>

### Hadoop Ecosystem Requirements

The following table summarizes the required Hadoop ecosystem components for your platform:

<table>
<thead>
<tr>
<th>Hadoop platform</th>
<th>Linux</th>
<th>Hadoop</th>
<th>HBase</th>
<th>ZooKeeper</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDH 5.16.1, CDH 5.14.2, CDH 5.14.0, CDH 5.13.3, CDH 5.13.2, CDH 5.12.0</td>
<td>CentOS/RHEL 6</td>
<td>2.6.0</td>
<td>1.2.0</td>
<td>3.4.5</td>
</tr>
<tr>
<td>HDP 2.6.4, HDP 2.6.3, HDP 2.6.1</td>
<td>CentOS/RHEL 6</td>
<td>2.7.1</td>
<td>1.1.2</td>
<td>3.4.5</td>
</tr>
</tbody>
</table>
Linux Configuration Requirements
The following table summarizes Linux configuration requirements for running Splice Machine on your cluster:

<table>
<thead>
<tr>
<th>Configuration Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure SSH access:</td>
<td>Configure the user account that you're using for cluster administration for password-free access, to simplify installation.</td>
</tr>
<tr>
<td>Configure swappiness:</td>
<td>echo 'vm.swappiness = 0' &gt;&gt; /etc/sysctl.conf</td>
</tr>
<tr>
<td>If you are using Ubuntu:</td>
<td>rm /bin/sh ; ln -sf /bin/bash /bin/sh</td>
</tr>
<tr>
<td>If your using CentOS or RHEL:</td>
<td>sed -i '/requiretty/ s/^/#/' /etc/sudoers</td>
</tr>
<tr>
<td>Configuration Step</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Required software:</td>
<td>Verify that the following set of software (or packages) is available on each node in your cluster:</td>
</tr>
<tr>
<td></td>
<td>★ curl</td>
</tr>
<tr>
<td></td>
<td>★ Oracle JDK 1.8, update 60 or higher. We recommend against using JDK 1.8.0_40 or OpenJDK.</td>
</tr>
<tr>
<td></td>
<td>★ NOTE: Your platform management software may re-install JDK during its own installation process.</td>
</tr>
<tr>
<td></td>
<td>★ nscd</td>
</tr>
<tr>
<td></td>
<td>★ ntp</td>
</tr>
<tr>
<td></td>
<td>★ openssh, openssh-clients, and openssh-server</td>
</tr>
<tr>
<td></td>
<td>★ patch</td>
</tr>
<tr>
<td></td>
<td>★ rlwrap</td>
</tr>
<tr>
<td></td>
<td>★ wget</td>
</tr>
<tr>
<td>Additional required software on CentOS or</td>
<td>If you’re running on CENTOS or RHEL, you also need to have this software available on each node:</td>
</tr>
<tr>
<td>RHEL</td>
<td>★ ftp</td>
</tr>
<tr>
<td></td>
<td>★ EPEL repository</td>
</tr>
<tr>
<td>Services that must be started</td>
<td>You need to make sure that the following services are enabled and started:</td>
</tr>
<tr>
<td></td>
<td>★ nscd</td>
</tr>
<tr>
<td></td>
<td>★ ntpd (ntp package)</td>
</tr>
<tr>
<td></td>
<td>★ ss hd (openssh-server package)</td>
</tr>
<tr>
<td>Time zone setting</td>
<td><strong>Make sure all nodes in your cluster are set to the same time zone.</strong></td>
</tr>
</tbody>
</table>

**Standalone Version Prerequisites**

You can use the standalone version of Splice Machine on MacOS and Linux computers that meet these basic requirements:
### Component Requirements

<table>
<thead>
<tr>
<th>Component</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operating System</strong></td>
<td>Mac OS X, version 10.8 or later.</td>
</tr>
<tr>
<td></td>
<td>CentOS 6.4 or equivalent.</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>Splice Machine recommends 2 or more multiple-core CPUs.</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>At least 16 GB RAM, of which at least 10 GB is available.</td>
</tr>
<tr>
<td><strong>Disk Space</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Software</strong></td>
<td>You must have JDK installed on your computer.</td>
</tr>
</tbody>
</table>

**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.
# Splice Machine Enterprise Edition Support

The following table summarizes support for the Enterprise Edition of Splice Machine.

<table>
<thead>
<tr>
<th><strong>Support Summary</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Term</strong></td>
</tr>
<tr>
<td><strong>Named Support Contacts</strong></td>
</tr>
<tr>
<td><strong>Access Channels</strong></td>
</tr>
<tr>
<td><strong>Support Hours</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Support Response Times</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Severity 1</strong></td>
</tr>
<tr>
<td><strong>Severity 2</strong></td>
</tr>
<tr>
<td><strong>Severity 3</strong></td>
</tr>
<tr>
<td><strong>Severity 4</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Severity Definitions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Severity 1</strong></td>
</tr>
<tr>
<td><strong>Severity 2</strong></td>
</tr>
<tr>
<td><strong>Severity 3</strong></td>
</tr>
<tr>
<td><strong>Severity 4</strong></td>
</tr>
</tbody>
</table>
Splice Machine Installation Guide

Splice Machine offers these product versions:

» You can install our *On-Premise Database* product for use on Cloudera, Hortonworks, and MapR platform managers. You'll find a link to the installer and instructions for your platform and product version in our Installer Links page, in this section.

» You can also install a *standalone version* of Splice Machine on your Mac OS or CentOS computers. Links and instructions for the standalone version are also found in our Installer Links page.

» Our *Database-as-Service* product runs in the cloud and is managed for you by Splice Machine; no installation is required.
Splice Machine Requirements

This topic summarizes the hardware and software requirements for Splice Machine running on a cluster or on a standalone computer, in these sections:

- **Cluster Node Requirements** summarizes requirements for running Splice Machine on a cluster, including these subsections:
  - Amazon Web Services (AWS) Requirements
  - Hadoop Ecosystem Requirements
  - Linux Configuration Requirements
- **Standalone Version Prerequisites** summarizes requirements for running the standalone version of Splice Machine.
- **Java JDK Requirements** summarizes which Java JDK requirements for running Splice Machine.

This is an On-Premise-Only topic! Learn more

Cluster Node Requirements

The following table summarizes the minimum requirements for the nodes in your cluster:

<table>
<thead>
<tr>
<th>Component</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cores</strong></td>
<td>Splice Machine recommends that each node in your cluster have 8-12 hyper-threaded cores (16-32 hyper-threads) for optimum throughput and concurrency.</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>We recommend that each machine in your cluster have at least 64 GB of available memory.</td>
</tr>
<tr>
<td><strong>Disk Space</strong></td>
<td>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 ext4 file system.</td>
</tr>
<tr>
<td><strong>Hadoop Ecosystem</strong></td>
<td>The table in the next section, Hadoop Ecosystem Requirements, summarizes the specific Hadoop component versions that we support in each of our product releases.</td>
</tr>
</tbody>
</table>
Amazon Web Services (AWS) Requirements
If you're running on AWS, your cluster must meet these minimum requirements:

<table>
<thead>
<tr>
<th>Component</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Cluster Size</td>
<td>The minimum cluster size on AWS is 5 nodes:</td>
</tr>
<tr>
<td></td>
<td>• 1 master node</td>
</tr>
<tr>
<td></td>
<td>• 4 worker nodes</td>
</tr>
<tr>
<td>Minimum Node Size</td>
<td>Minimum recommended size of each node is m4.4xlarge.</td>
</tr>
<tr>
<td>Disk Space</td>
<td>Minimum recommended storage space:</td>
</tr>
<tr>
<td></td>
<td>• 100 GB EBS root drive</td>
</tr>
<tr>
<td></td>
<td>• 4 EBS data drives per node</td>
</tr>
<tr>
<td></td>
<td>Note that the required number of data drives per node depends on your use case.</td>
</tr>
</tbody>
</table>

Hadoop Ecosystem Requirements
The following table summarizes the required Hadoop ecosystem components for your platform:

<table>
<thead>
<tr>
<th>Hadoop platform</th>
<th>Linux</th>
<th>Hadoop</th>
<th>HBase</th>
<th>ZooKeeper</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RHEL 6</td>
<td>2.6.0</td>
<td>1.2.0</td>
<td>3.4.5</td>
</tr>
<tr>
<td>HDP 2.6.4, HDP 2.6.3, HDP 2.6.1</td>
<td>CentOS/</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RHEL 6</td>
<td>2.7.1</td>
<td>1.1.2</td>
<td>3.4.5</td>
</tr>
</tbody>
</table>
Linux Configuration Requirements
The following table summarizes Linux configuration requirements for running Splice Machine on your cluster:

<table>
<thead>
<tr>
<th>Configuration Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure SSH access:</td>
<td>Configure the user account that you're using for cluster administration for</td>
</tr>
<tr>
<td></td>
<td>password-free access, to simplify installation.</td>
</tr>
<tr>
<td>Configure swappiness:</td>
<td>echo 'vm.swappiness = 0' &gt;&gt; /etc/sysctl.conf</td>
</tr>
<tr>
<td>If you are using Ubuntu:</td>
<td>rm /bin/sh ; ln -sf /bin/bash /bin/sh</td>
</tr>
<tr>
<td>If your using CentOS or RHEL:</td>
<td>sed -i '/requiretty/ s/^/#/' /etc/sudoers</td>
</tr>
</tbody>
</table>
### Configuration Step

<table>
<thead>
<tr>
<th>Configuration Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required software:</td>
<td>Verify that the following set of software (or packages) is available on each node in your cluster:</td>
</tr>
<tr>
<td></td>
<td>- curl</td>
</tr>
<tr>
<td></td>
<td>- Oracle JDK 1.8, update 60 or higher. We recommend against using JDK 1.8.0_40 or OpenJDK.</td>
</tr>
<tr>
<td></td>
<td>- nscd</td>
</tr>
<tr>
<td></td>
<td>- ntp</td>
</tr>
<tr>
<td></td>
<td>- openssh, openssh-clients, and openssh-server</td>
</tr>
<tr>
<td></td>
<td>- patch</td>
</tr>
<tr>
<td></td>
<td>- rlwrap</td>
</tr>
<tr>
<td></td>
<td>- wget</td>
</tr>
<tr>
<td>Additional required software on CentOS or RHEL</td>
<td>If you’re running on CENTOS or RHEL, you also need to have this software available on each node:</td>
</tr>
<tr>
<td></td>
<td>- ftp</td>
</tr>
<tr>
<td></td>
<td>- EPEL repository</td>
</tr>
<tr>
<td>Services that must be started</td>
<td>You need to make sure that the following services are enabled and started:</td>
</tr>
<tr>
<td></td>
<td>- nscd</td>
</tr>
<tr>
<td></td>
<td>- ntpd (ntpd package)</td>
</tr>
<tr>
<td></td>
<td>- sshd (openssh-server package)</td>
</tr>
<tr>
<td>Time zone setting</td>
<td>Make sure all nodes in your cluster are set to the same time zone.</td>
</tr>
</tbody>
</table>

### Standalone Version Prerequisites

You can use the standalone version of Splice Machine on MacOS and Linux computers that meet these basic requirements:
<table>
<thead>
<tr>
<th>Component</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Mac OS X, version 10.8 or later.</td>
</tr>
<tr>
<td></td>
<td>CentOS 6.4 or equivalent.</td>
</tr>
<tr>
<td>CPU</td>
<td>Splice Machine recommends 2 or more multiple-core CPUs.</td>
</tr>
<tr>
<td>Memory</td>
<td>At least 16 GB RAM, of which at least 10 GB is available.</td>
</tr>
<tr>
<td>Disk Space</td>
<td>At least 100 GB of disk space available for Splice Machine software, plus as much 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.</td>
</tr>
<tr>
<td>Software</td>
<td>You must have JDK installed on your computer.</td>
</tr>
</tbody>
</table>

**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.
# Splice Machine Installation Guide

The table below links to instructions for installing Splice Machine. We recommend that you review the installation requirements for Splice Machine before proceeding.

[This is an On-Premise-Only topic! Learn more]

<table>
<thead>
<tr>
<th>Splice Machine Version</th>
<th>Platform Version</th>
<th>Install Instructions URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>CDH 5.16.1</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/cdh5.16.1/docs/CDH-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/cdh5.16.1/docs/CDH-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>CDH 5.12.0</td>
<td><a href="https://github.com/splicemachine/spliceengine/tree/branch-2.7/platforms/cdh5.12.0/docs/CDH-installation.md">https://github.com/splicemachine/spliceengine/tree/branch-2.7/platforms/cdh5.12.0/docs/CDH-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>HDP 2.6.4</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.6.4/docs/HDP-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.6.4/docs/HDP-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>HDP 2.6.3</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.6.3/docs/HDP-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.6.3/docs/HDP-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>HDP 2.6.1</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.6.1/docs/HDP-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/hdp2.6.1/docs/HDP-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>MapR 5.2.0</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/mapr5.2.0/docs/MapR-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/mapr5.2.0/docs/MapR-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>Standalone</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/std/docs/STD-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.7/platforms/std/docs/STD-installation.md</a></td>
</tr>
</tbody>
</table>

See our Import and Query Sample Standalone Sample Data topic if you wish to import and query the sample data packaged with our standalone installer.
<table>
<thead>
<tr>
<th>Splice Machine Version</th>
<th>Platform Version</th>
<th>Install Instructions URL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CDH 5.12.0</td>
<td><a href="https://github.com/splicemachine/spliceengine/tree/branch-2.5/platforms/cdh5.12.0/docs/CDH-installation.md">https://github.com/splicemachine/spliceengine/tree/branch-2.5/platforms/cdh5.12.0/docs/CDH-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>HDP 2.6.4</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/hdp2.6.4/docs/HDP-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/hdp2.6.4/docs/HDP-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>HDP 2.6.3</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/hdp2.6.3/docs/HDP-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/hdp2.6.3/docs/HDP-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>HDP 2.6.1</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/hdp2.6.1/docs/HDP-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/hdp2.6.1/docs/HDP-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>HDP 2.5.5</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/hdp2.5.5/docs/HDP-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/hdp2.5.5/docs/HDP-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>MapR 5.2.0</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/mapr5.2.0/docs/MapR-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/mapr5.2.0/docs/MapR-installation.md</a></td>
</tr>
<tr>
<td></td>
<td>Standalone</td>
<td><a href="https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/stdlib/docs/STD-installation.md">https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms/stdlib/docs/STD-installation.md</a></td>
</tr>
</tbody>
</table>

See our Import and Query Sample Standalone Sample Data topic if you wish to import and query the sample data packaged with our standalone installer.

For access to the source code for the Community Edition of Splice Machine, visit [our open source GitHub repository](https://github.com/splicemachine/spliceengine/blob/branch-2.5/platforms).
How to Install Splice Machine for Zeppelin

This topic describes how to install the Splice Machine interpreter for Apache Zeppelin.

If you’ve not yet installed Zeppelin, please follow the official Apache instructions for the latest version. At the time of this writing, the latest version is 0.8.0, and the installation instructions are here:

http://zeppelin.apache.org/docs/0.8.0/usage/interpreter/installation.html

Add the Splice Machine Zeppelin Interpreter

To add the Splice Machine Zeppelin interpreter to the set of interpreters available in your Zeppelin environment, follow these steps:

1. Download the `splicemachine-zeppelin-0.8.0.jar` file. We recommend saving the download in the `/tmp` directory, though you can use another directory if you add it to the Zeppelin path.

2. Navigate to your `zeppelin` home directory.

3. Use the following command to install the Splice Machine interpreter for Zeppelin:

   ```bash
   .bin//install-interpreter.sh --name splicemachine --artifact /tmp/splicemachine-zeppelin-0.8.0.jar
   ```

   If you saved the jar file to a directory other than `/tmp`, remember to change its location in the install command.

4. Restart the Zeppelin server

5. Navigate to the Interpreter page in Zeppelin, which is accessible from drop-down menu in the upper-right corner of the Zeppelin window.

6. Click the Create button in the Interpreter window. This opens a form; enter the following into the form fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value to Enter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>splicemachine</td>
</tr>
<tr>
<td>Interpreter Group</td>
<td>Select splicemachine from the drop-down menu.</td>
</tr>
</tbody>
</table>
### Field | Value to Enter
---|---
default.user | Set this value to your default user name; this value is initially set to splice.
default.password | Set this value to your default password.
default.url | Set this value to match your server's configuration; this value is initially set to jdbc:splice://localhost:1527/splicedb.
default.splitQueries | true

7. Save your changes.

### Restart Zeppelin

After installing the Splice Machine interpreter, you need to restart Zeppelin again:

1. Stop Zeppelin:
   
   ```
   ./bin/zeppelin-daemon.sh stop
   ```

2. Start Zeppelin:
   
   ```
   ./bin/zeppelin-daemon.sh start
   ```

### Start Using Splice Machine in Your Zeppelin Notebooks

To use the new interpreter in a notebook, you must bind it into the notebook, following these steps:

1. Open or create a notebook in Zeppelin.
2. Click the gear icon in the upper right corner of the Zeppelin window to display the list of interpreters that are currently bound to this notebook:
3. Scroll down through the list of interpreters, and click the splicemachine button:

4. Click the Save button to bind the interpreter to this notebook.

If you're able to connect to a Splice Machine instance, you can create paragraphs in your Zeppelin notebooks that work with your database, assuming: specify the `%splicemachine` interpreter at the top of the paragraph, and issue commands just as you would with the splice command line interpreter. For example:

```sql
%splicemachine
drop table myTable;
create table myTable ( id int, name varchar(64) );
insert into myTable value (1, 'John Doe'), (2, 'Jane Doe');
select * from myTable where id > 10;
```
Importing and Querying Standalone Demo Data

The standalone version of Splice Machine includes sample data that you can use to quickly try importing and querying meaningful data. This topic walks you through using the sample data.

You must install the standalone version of Splice Machine on your computer before following the steps in this tutorial. You’ll find the link to the standalone installer and instructions in our Installing Splice Machine page.

The Splice Machine installer package includes demo data that you can import into your database, so you can get used to working with your new database. We recommend that you follow the steps in this topic to import this demo data, and then run a few test queries against it to verify your installation.

About the Demo Data

The demo data included in your installer package requires about 30 MB in compressed format. Importing the demo data creates three tables, each of which contains one million records:

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_HEADER</td>
<td>Standard <em>headers</em> from a transaction system</td>
</tr>
<tr>
<td>T_DETAIL</td>
<td>Standard <em>detail</em> records from a transaction system</td>
</tr>
<tr>
<td>CUSTOMERS</td>
<td>A list of target <em>customers</em></td>
</tr>
</tbody>
</table>

Follow the Written Instructions

Import the Data

Follow these steps to import the demo data into your Splice Machine database:

1. **Start the command line interpreter**
   You can use the Splice Machine command line interpreter (CLI), or `splice>` prompt, to work directly with your database. If you're using the cluster version of Splice Machine, you can access the `splice>` prompt by entering this shell command on any node on which it is available:

   ```
   ./sqlshell.sh
   ```
If you’re using the standalone version of Splice Machine, use these steps to access the `splice>` prompt:

```bash
cd <your.splicemachine-directory>
./bin/sqlshell.sh
```

2. **Modify the script that loads the data to use your path:**

Before running the `loadall.sql` script, you must change the file path used in the script.

There are calls to `SYSCS_UTIL.IMPORT_DATA` near the bottom of the script. Change the file path parameter in each of these calls to use the absolute path to your Splice Machine demodata directory:

```bash
call SYCS_UTIL.IMPORT_DATA('SPLICE', 'T_HEADER', null, '<yourPath>/demodata/data/theader.csv', ...;
call SYCS_UTIL.IMPORT_DATA('SPLICE', 'T_DETAIL', null, '<yourPath>/demodata/data/tdetail.csv', ...;
call SYCS_UTIL.IMPORT_DATA('SPLICE', 'CUSTOMERS', null, '<yourPath>/demodata/data/customers.csv', ...;
```

Make sure you use the absolute (versus relative) path. For example:

```bash
call SYCS_UTIL.IMPORT_DATA('SPLICE', 'T_HEADER', null, '/Users/myName/mySplice/demodata/data/theader.csv', ...;
call SYCS_UTIL.IMPORT_DATA('SPLICE', 'T_DETAIL', null, '/Users/myName/mySplice/demodata/data/tdetail.csv', ...;
call SYCS_UTIL.IMPORT_DATA('SPLICE', 'CUSTOMERS', null, '/Users/myName/mySplice/demodata/data/customers.csv', ...;
```

3. **Run the modify script to loads the data:**

From the `splice>` prompt, run the file that will load the data, using single quotes around the path/filename (and remember to include the semicolon at the end):

```bash
splice> run 'demodata/sql/loadall.sql';
```

4. **Wait for the script to finish**

If your database is not currently running, start it up and launch the command line interpreter (`splice>` prompt) by issuing this command in your terminal window:

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

The loading process can take several minutes: the `loadall.sql` file creates the schema, loads the data, and creates indexes for the tables.

**NOTE:** While the database is running, logging information is written to the `splice.log` file, which is found in the `splicemachine` directory.

When you again see the `splice>` prompt, the demo data is ready to use. We recommend running the sample queries in the next section to get a feel for using Splice Machine and the demo data.
Run Sample Queries
After you have imported the demo data, you can use the `splice>` command line interpreter to run the sample queries on this page to get some experience with using Splice Machine.

You can simply copy the select command from each of the samples below to your clipboard. Then paste from the clipboard at the `splice>` prompt and press the `Enter` key or `Return` key to submit the query.

**Example of Selecting a Subset**
You can use the following query to select the customer IDs from a subset of the transaction detail (T_DETAIL) table, based on transaction date and category ID.

```sql
select customer_master_id
from T_DETAIL d
where TRANSACTION_DT >= DATE('2010-01-01')
  and TRANSACTION_DT <= DATE('2013-12-31')
  AND ORIGINALSKUCATEGORY_ID >= 44427
  and original_sku_category_id <= 44431;
```

**Example of Selecting With a Join**
You can use the following to query a join of the T_HEADER and CUSTOMERS tables.

```sql
select t.transaction_header_key, t.transaction_dt, t.store_nbr,
       t.geocapture_flg, t.exchange_rate_percent
from T_HEADER t, CUSTOMERS c
where c.customer_master_id = t.customer_master_id
  and t.customer_master_id > 14000
  and t.customer_master_id < 15000;
```

Troubleshooting Transaction Exceptions on MacOS
If you’re running transactions in the standalone version of Splice Machine on MacOS, you may run into an exception caused by the clock having moved backwards. This happens only rarely, and is due to the fact that OS X has its own time-maintenance daemon that can (rarely) cause the clock to move backwards, which causes a transaction exception.

When this happens, you’ll see an exception messages like the following:

```
SQLSTATE: XJ001Java exception: 'java.io.IOException: java.lang.IllegalStateException: Unable to obtain timestamp, clock moved backwards
```

To correct the problem, simply re-run the query or statement that generated the exception.
**Splice Machine Administrator’s Guide**

This *Administrator’s Guide* describes the administrative tasks associated with configuring and maintaining your Splice Machine software, in the topics shown in the following table.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting Your Database</td>
<td>How to start or restart your database.</td>
</tr>
<tr>
<td>Backing Up and Restoring</td>
<td>How to back up and restore your Splice Machine database.</td>
</tr>
<tr>
<td>Cleaning Your Database</td>
<td>How to clean your Splice Machine database.</td>
</tr>
<tr>
<td>Shutting Down Your Database</td>
<td>How to shut down your Splice Machine database.</td>
</tr>
<tr>
<td>Accessing Derby Properties</td>
<td>How to enable and disable Derby features by setting Derby properties.</td>
</tr>
</tbody>
</table>

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

- 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

This is an On-Premise-Only topic! Learn more

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 splicemachin
   ```
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:
3. **Run the Splice Machine shut-down script:**
   
   ```
   cd splicemachine
   $ ./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; contact Splice Machine Sales for information.]

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
- restore your database from a backup
- validate backups
- 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 and Restore Types and Compatibility

To back up and restore your entire database, use these system procedures:

- SYCS_UTIL.SYCS_BACKUP_DATABASE
- SYCS_UTIL.SYCS_RESTORE_DATABASE

If you only want to back up or restore the tables and indexes belonging to a specific schema, you can use these procedures:

- SYCS_UTIL.SYCS_BACKUP_SCHEMA
- SYCS_UTIL.SYCS_RESTORE_SCHEMA

And if you only want to back up or restore a specific table, you can use these procedures:

- SYCS_UTIL.SYCS_BACKUP_TABLE
- SYCS_UTIL.SYCS_RESTORE_TABLE
Note that you can only use specific restore procedures with specific types of backups. For example, you can use the `RESTORE_TABLE` procedure to restore from a backup created by `BACKUP_TABLE`, but you cannot use `RESTORE_TABLE` to restore from a backup created by `BACKUP_DATABASE`. The following table summarizes backup-restore compatibility:

<table>
<thead>
<tr>
<th>If you backed up with this procedure:</th>
<th>You can restore with these procedures:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCS_UTIL.SYSCS_BACKUP_DATABASE</td>
<td>SYSCS_UTIL.SYSCS_RESTORE_DATABASE</td>
</tr>
<tr>
<td>SYSCS_UTIL.SYSCS_BACKUP_SCHEMA</td>
<td>SYSCS_UTIL.SYSCS_RESTORE_SCHEMA,</td>
</tr>
<tr>
<td></td>
<td>SYSCS_UTIL.SYSCS_RESTORE_TABLE</td>
</tr>
<tr>
<td>SYSCS_UTIL.SYSCS_BACKUP_TABLE</td>
<td>SYSCS_UTIL.SYSCS_RESTORE_TABLE</td>
</tr>
</tbody>
</table>

**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 Troubleshooting Backups section of our *Best Practices 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. To use incremental backups, you **must** make a few HBase configuration changes and be aware of one significant restriction, as described below, in the Incremental Backup Configuration and Limitations section.

Because backups can consume a lot of disk space, most customers define a backup strategy that blends their needs for security, recover-ability, and space restrictions. Since incremental backups require a lot less space than do full backups, and allow for faster recovery of data, many customers choose to schedule frequent 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

<table>
<thead>
<tr>
<th>System Table</th>
<th>Contains Information About</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS.SYSBACKUP</td>
<td>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.</td>
</tr>
<tr>
<td>SYS.SYSBACKUPITEMS</td>
<td>Each item (table) in a backup.</td>
</tr>
</tbody>
</table>

### 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.

### Incremental Backup Configuration and Limitations

If you’re going to perform incremental backups, you **must** follow these steps:

1. Add the following options to your `hbase-site.xml` configuration file:
   ```properties
   hbase.master.hfilecleaner.plugins = com.splicemachine.hbase.SpliceHFileCleaner,
   org.apache.hadoop.hbase.master.cleaner.TimeToLiveHFileCleaner
   ```
2. Purge the `/hbase/backup/data` directory.
3. Run a full backup.
4. Run your first incremental backup.

### Using the Backup Operations

This section summarizes and provides examples of using the Splice Machine backup operations:

- [Running an Immediate Backup](#)
- [Restoring Your Database From a Previous Backup](#)
Validating Backups
Reviewing Backup Information
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.

---

**Running an Immediate Backup**

Use the `SYSCS_UTIL.SYSCS_BACKUP_DATABASE` system procedure to immediately run a full or incremental backup.

```
SYCS_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 SYCS_UTIL.SYCS_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 SYCS_UTIL.SYCS_BACKUP_DATABASE( '/home/backup', 'incremental' );
```
Restoring Your Database From a Previous Backup

To restore your database from a previous backup, use the `SYSCS_UTIL.SYSCS_RESTORE_DATABASE` system procedure:

```sql
SYSCS_UTIL.SYSCS_RESTORE_DATABASE(backupDir, backupId, validate);
```

**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.

**validate**

A `BOOLEAN` value that specifies whether you want the backup validated before restoring the database from it. If this is `true` and the validation finds inconsistencies, the database is not restored. If this is `false`, validation is not performed.

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.
- The restore runs asynchronously, which means that you **need to** look at the region server log for a message that the restore is complete, and then reboot 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`, after first validating the backup:

```sql
call SYSCS_UTIL.SYSCS_RESTORE_DATABASE('/home/backup', 1273, true);
```

**Validating Backups**

You can validate a database backup with the `SYSCS_UTIL.VALIDATE_BACKUP` system procedure:

```sql
SYSCS_UTIL.SYSCS_VALIDATE_BACKUP(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 validate your database.

**Example: Validating a backup**
This example validates the backup stored in the /home/backup directory that has backupId=1273:

```sql
call SYSCS_UTIL.VALIDATE_BACKUP('/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 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 |FILESYSTEM
------------------------------------------------------
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
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 Backup That's In Progress
You can call the `SYSCS_UTIL.SYSCSANCEL_BACKUP` system procedure to cancel a backup that is currently running:

```sql
SYSCS_UTIL.SYSCSANCEL_BACKUP();
```

**Example: Cancel a running backup**
This example cancels the Splice Machine backup job that is currently running.

```sql
call SYSCS_UTIL.SYSCSCANCEL_BACKUP();
```

Deleting a Backup
Use the `SYSCS_UTIL.SYSCSDELETE_BACKUP` system procedure to delete a single backup:

```sql
SYSCS_UTIL.SYSCSDELETE_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`:

```sql
call SYSCS_UTIL.SYSCSDELETE_BACKUP(1273);
```

Deleting Outdated Backups
Use the `SYSCS_UTIL.SYSCSDELETE_OLD_BACKUPS` system procedure to delete all backups that are older than a certain number of days.

```sql
SYSCS_UTIL.SYSCSDELETE_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:

```sql
call SYSCS_UTIL.SYSCSDELETE_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:

- [https://cwiki.apache.org/confluence/display/HADOOP2/AmazonS3](https://cwiki.apache.org/confluence/display/HADOOP2/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
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:
4. **Use the** `hadoop` **command to view your configuration changes:**

```
$ hadoop conf | grep fs\s\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_ACCESS_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_BACKUP_SCHEMA`
» **SYSCS_UTIL.SYSCS_BACKUP_TABLE**
» **SYSCS_UTIL.SYSCS_CANCEL_BACKUP**
» **SYSCS_UTIL.SYSCS_DELETE_BACKUP**
» **SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS**
» **SYSCS_UTIL.SYSCS_RESTORE_DATABASE**
» **SYSCS_UTIL.SYSCS_RESTORE_SCHEMA**
» **SYSCS_UTIL.SYSCS_RESTORE_TABLE**
» **SYSCS_UTIL.VALIDATE_BACKUP**
» **SYSBACKUP** system table
» **SYSBACKUPITEMS** system table
Cleaning Your Database

Cleaning your database essentially wipes out any user-defined tables, indexes, and related items.

This is an On-Premise-Only topic! Learn more

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:

   ```
   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:
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 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

This is an On-Premise-Only topic! Learn more

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 topic describes how to enable and disable Derby features by setting Derby properties, which are divided into categories, as shown in the following table.

---

**This is an On-Premise-Only topic!** Learn more

---

**About Property Categories**

The following table summarizes the categories of Splice Machine properties.

<table>
<thead>
<tr>
<th>Property Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JVM (System)</strong></td>
<td>These properties can be as command line arguments to JVM:</td>
</tr>
<tr>
<td></td>
<td>- For Maven, include the command line arguments as an <code>&lt;argument&gt;</code> element in the pom.xml file. For example:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;argument&gt;-Dderby.language.logStatementText=true&lt;/argument&gt;</code></td>
</tr>
<tr>
<td></td>
<td>- For shell scripts, you may need to manually add the argument to the java executable command line. For example:</td>
</tr>
<tr>
<td></td>
<td><code>java -Dderby.language.logStatementText=true ...</code></td>
</tr>
</tbody>
</table>

System properties can also be set manually using the `System.setProperty(key, value)` function.
<table>
<thead>
<tr>
<th>Property Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Service**   | 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:  
  - Derby stores service properties in the `service.properties` file  
  - Splice Machine stores service properties in a Zookeeper element.  
You can temporarily change service properties with the `SYSCS_UTIL.SYSCS_SET_GLOBAL_DATABASE_PROPERTY` built-in system procedure. These changes will be lost when the server is next restarted.  
To make a permanent change in a Derby service property, modify the value in the `service.properties` file and then restart the server to apply the changes.  
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. |
| **Database**  | Splice Machine database properties are saved in a hidden HBASE table with `CONGLOMERATEID=16`. |
| **App (Derby/Splice)** | App properties for both Derby and Splice Machine are saved to the `derby.properties` file in the Derby/Splice home directory.  
App properties can also be saved to these HBASE XML configuration files:  
  - `hbase-default.xml`  
  - `hbase-site.xml`  
  - `splice-site.xml`  
Note that the XML files must reside in the CLASSPATH. |

When a property value is used, the property is searched for in the order shown in the table: the property is first 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.

**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();

<table>
<thead>
<tr>
<th>KEY</th>
<th>VALUE</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>derby.authentication.builtin.algorithm</td>
<td>SHA-256</td>
<td>DATABASE</td>
</tr>
<tr>
<td>derby.user.BROWSE</td>
<td>Browse</td>
<td>APP</td>
</tr>
<tr>
<td>derby.engineType</td>
<td>2</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derby.david.foo</td>
<td>WINTERS</td>
<td>APP</td>
</tr>
<tr>
<td>derby.connection.requireAuthentication</td>
<td>false</td>
<td>JVM</td>
</tr>
<tr>
<td>derby.locks.escalationThreshold</td>
<td>500</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derby.database.defaultConnectionMode</td>
<td>fullAccess</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derby.database.propertiesOnly</td>
<td>false</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derby.database.collation</td>
<td>UCS_BASIC</td>
<td>DATABASE</td>
</tr>
<tr>
<td>derby.language.logStatementText</td>
<td>false</td>
<td>JVM</td>
</tr>
<tr>
<td>derby.storage.propertiesId</td>
<td>16</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derby.language.logQueryPlan</td>
<td>true</td>
<td>JVM</td>
</tr>
<tr>
<td>splice.updateSystemProcs</td>
<td>false</td>
<td>JVM</td>
</tr>
<tr>
<td>derby.storage.rowLocking</td>
<td>false</td>
<td>SERVICE</td>
</tr>
</tbody>
</table>

## See Also

- [SYCS_UTIL.SYCS_SET_GLOBAL_DATABASE_PROPERTY](#)
- The *Derby Properties Guide* on the [Apache Derby documentation site](#)
Enabling Enterprise Features in Splice Machine

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.

This is an On-Premise-Only topic! Learn more

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 `SYSCS_UTIL.SYSCS_ENABLE_ENTERPRISE` system procedure to unlock those features. You only need to do this once:

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:

2. Edit the hbase-site.xml configuration file, adding this property:

```xml
<property>
  <name>splicemachine.enterprise.key</name>
  <value><your-Spli
de-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:

```
-Dsplicemach
```
Release Notes for the Splice Machine On-Premise Product

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 underlies this product are found in the individual release note pages for each major and patch release of the database.

### After Updating

After updating to a new release of Splice Machine, you may need to update your stored statement metadata by calling these system procedures:

```sql
CALL SYCS_UTIL.SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE();
CALL SYCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS();
CALL SYCS_UTIL.SYSCS_UPDATE_METADATA_STORED_STATEMENTS();
```

### Supported Platforms

The supported platforms for release 2.7 are:

- MapR 5.2.0
- HortonWorks HDP2.6.4, 2.6.3, 2.5.5

### Enterprise-only Features


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 Commands Reference

This guide contains reference information for using the Splice Machine command line interpreter, which is also known as the Splice Prompt (splice>).

The Using section of the guide will help you to get started, in the topics:

- Getting Started with the CLI introduces you to using the CLI.
- Command Line Syntax summarizes command line parameters and syntax.
- Scripting the CLI shows you how to script a set of commands to submit via the CLI.
- Using RLWrap with splice> summarizes the RLWrap commands you can use to enhance the CLI.

The remainder of this guide contains a reference topic page for each Splice Machine command. As shown in the tables below, you can use many of the commands when connected in any way (including JDBC and ODBC) to Splice Machine, while some commands can only be used via our command line interpreter.

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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyze</td>
<td>Collects statistics for a table or schema.</td>
<td>splice&gt; analyze table myTable;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>splice&gt; analyze schema myschema;</td>
</tr>
<tr>
<td>Autocommit</td>
<td>Turns the connection's auto-commit mode on or off.</td>
<td>splice&gt; autocommit off;</td>
</tr>
<tr>
<td>Export</td>
<td>Exports query results to binary files.</td>
<td>splice&gt; EXPORT_BINARY('/my/export/dir', true, 'parquet') SELECT a,b,sqrt(c) FROM t1 WHERE a &gt; 100;</td>
</tr>
<tr>
<td>Commit</td>
<td>Commits the currently active transaction and initiates a new transaction.</td>
<td>splice&gt; commit;</td>
</tr>
<tr>
<td>Execute</td>
<td>Executes an SQL prepared statement or SQL command string.</td>
<td>splice&gt; execute 'insert into myTable(id, val) values(?,?)' ;</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Usage</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Explain</td>
<td>Displays the execution plan for an SQL statement.</td>
<td>splice&gt; explain select count(*) from si;</td>
</tr>
<tr>
<td>Export</td>
<td>Exports query results to CSV files.</td>
<td>splice&gt; EXPORT('/my/export/dir', null, null, null, null, null, null) SELECT a,b,sqrt(c) FROM join t2 on t1.a=t2.a;</td>
</tr>
<tr>
<td>Prepare</td>
<td>Creates a prepared statement for use by other commands.</td>
<td>splice&gt; prepare seeMenu as 'SELECT * FROM menu';</td>
</tr>
<tr>
<td>Release</td>
<td>Releases a savepoint.</td>
<td>splice&gt; release savepoint gSavePt1;</td>
</tr>
<tr>
<td>Savepoint</td>
<td>Removes a previously prepared statement.</td>
<td>splice&gt; remove seeMenu;</td>
</tr>
<tr>
<td>Rollback</td>
<td>Rolls back the currently active transaction and initiates a new transaction.</td>
<td>splice&gt; rollback;</td>
</tr>
<tr>
<td>Rollback</td>
<td>Rolls the current transaction back to the specified savepoint.</td>
<td>splice&gt; rollback to savepoint gSavePt1;</td>
</tr>
<tr>
<td>Savepoint</td>
<td>Creates a savepoint within the current transaction.</td>
<td>splice&gt; savepoint gSavePt1;</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Connect to a database via its URL.</td>
<td>splice&gt; connect 'jdbc:splice://xyz:1527/splicedb';</td>
</tr>
<tr>
<td>Describe</td>
<td>Displays a description of a table or view.</td>
<td>splice&gt; describe myTable;</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Usage</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Disconnects from a database.</td>
<td>splice&gt; disconnect</td>
</tr>
<tr>
<td>Elapsedtime</td>
<td>Enables or disables display of elapsed time for command execution.</td>
<td>splice&gt; elapsedtime on;</td>
</tr>
<tr>
<td>Exit</td>
<td>Causes the command line interface to exit.</td>
<td>splice&gt; exit;</td>
</tr>
<tr>
<td>Help</td>
<td>Displays a list of the available commands.</td>
<td>splice&gt; help;</td>
</tr>
<tr>
<td>MaximumDisplayWidth</td>
<td>Sets the maximum displayed width for each column of results displayed by the command line interpreter.</td>
<td>splice&gt; maximumdisplaywidth 30;</td>
</tr>
<tr>
<td>Run</td>
<td>Runs commands from a file.</td>
<td>splice&gt; run myCmdFile;</td>
</tr>
<tr>
<td>Set Connection</td>
<td>Allows you to specify which connection is the current connection</td>
<td>splice&gt; set connection sample1;</td>
</tr>
<tr>
<td>Set Session_Property</td>
<td>Allows you to specify default hint values for certain query hints</td>
<td>splice&gt; set session_property useSpark=true;</td>
</tr>
<tr>
<td>Show Connections</td>
<td>Displays information about active connections and database objects.</td>
<td>splice&gt; show connections;</td>
</tr>
<tr>
<td>Show Create Table</td>
<td>command displays the DDL used with the <code>create table</code> statement to create a specified table.</td>
<td>splice&gt; show create table players;</td>
</tr>
<tr>
<td>Show Functions</td>
<td>Displays information about functions defined in the database or in a schema.</td>
<td>splice&gt; show functions in splice;</td>
</tr>
<tr>
<td>Show Indexes</td>
<td>Displays information about the indexes defined on a table, a database, or a schema.</td>
<td>splice&gt; show indexes from mytable;</td>
</tr>
<tr>
<td>Show Primary Keys</td>
<td>Displays information about the primary keys in a table.</td>
<td>splice&gt; show primarykeys from mySchema.myTable;</td>
</tr>
<tr>
<td>Show Procedures</td>
<td>Displays information about active connections and database objects.</td>
<td>splice&gt; show procedures in syscs_util;</td>
</tr>
<tr>
<td>Show Roles</td>
<td>Displays information about all of the roles defined in the database.</td>
<td>splice&gt; show roles;</td>
</tr>
<tr>
<td>Show Schemas</td>
<td>Displays information about the schemas in the current connection.</td>
<td>splice&gt; show schemas;</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Usage</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>Show Synonyms</td>
<td>Displays information about the synonyms that have been created in a database or schema.</td>
<td><code>splice&gt; show synonyms;</code></td>
</tr>
<tr>
<td>Show Tables</td>
<td>Displays information about all of the tables in a database or schema.</td>
<td><code>splice&gt; show tables in SPLICE;</code></td>
</tr>
<tr>
<td>Show Views</td>
<td>Displays information about all of the active views in a schema.</td>
<td><code>splice&gt; show views in SPLICE;</code></td>
</tr>
</tbody>
</table>
Getting Started With the splice> Command Line Interface

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.

Our Command Line Reference contains additional information about command line syntax and commands, and includes examples of each available command.

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
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 Shell
For help: "splice> help;" SPLICE** = current connection
splice>
```

`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:

<table>
<thead>
<tr>
<th>Step</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit splice&gt;</td>
<td><code>splice&gt; quit; (exit;)</code></td>
</tr>
<tr>
<td>Stop Splice Machine processes</td>
<td><code>$ ./bin/stop-splice.sh</code></td>
</tr>
<tr>
<td>Restart Splice Machine processes</td>
<td><code>$ ./bin/start-splice.sh</code></td>
</tr>
<tr>
<td>Restart splice&gt;</td>
<td><code>$ ./bin/sqlshell.sh</code></td>
</tr>
</tbody>
</table>

### 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=YourUserId;password=YourPassword' AS DEMO;

Anatomy of a Connection String
Here's how to breakdown the connection strings we use to connect to a database:

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

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

```
splice> Disconnect DEMO;
splice> show Connections;
SPLICE   - jdbc:splice://localhost:1527/splicedb
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/splicedb
* = 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_SCHEMA
-----------------------------
NULLID
SPLICE
SQLJ
SYS
SYSCAT
SYSCS_DIAG
SYSCS_UTIL
SYSFUN
SYSSIBM
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>
<thead>
<tr>
<th>TABLE_SCHM</th>
<th>TABLE_NAME</th>
<th>CONGLOM_ID</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>SYSALIASES</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSBACKUP</td>
<td>992</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSBACKUPITEMS</td>
<td>1104</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCHECKS</td>
<td>336</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCOLPERMS</td>
<td>608</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCOLUMNS</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCOLUMNSTATS</td>
<td>1264</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCONGLOMERATES</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCONSTRANTS</td>
<td>304</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSDEPENDS</td>
<td>368</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSFILES</td>
<td>288</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSFOREIGNKEYS</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLES</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSTATEMENTS</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLEPERMS</td>
<td>592</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTRIGGERS</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLESTATS</td>
<td>1296</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSCHEMAPERMS</td>
<td>1328</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSCHEMAS</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSCENENTES</td>
<td>864</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSSTATEMENTS</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLEPERMS</td>
<td>592</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLES</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLESTATS</td>
<td>1296</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTRIGGERS</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLES</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSTATEMENTS</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLEPERMS</td>
<td>592</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLES</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLESTATS</td>
<td>1296</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTRIGGERS</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLES</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLESTATS</td>
<td>1296</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTRIGGERS</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLES</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLESTATS</td>
<td>1296</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTRIGGERS</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLES</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLESTATS</td>
<td>1296</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTRIGGERS</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLES</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLESTATS</td>
<td>1296</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTRIGGERS</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLES</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLESTATS</td>
<td>1296</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTRIGGERS</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLES</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLESTATS</td>
<td>1296</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTRIGGERS</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLES</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTABLESTATS</td>
<td>1296</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTRIGGERS</td>
<td>576</td>
<td></td>
</tr>
</tbody>
</table>

34 rows selected

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

splice> show tables in SPLICE;

<table>
<thead>
<tr>
<th>TABLE_SCHM</th>
<th>TABLE_NAME</th>
<th>CONGLOM_ID</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPLICE</td>
<td>CUSTOMERS</td>
<td>1568</td>
<td></td>
</tr>
<tr>
<td>SPLICE</td>
<td>T_DETAIL</td>
<td>1552</td>
<td></td>
</tr>
<tr>
<td>SPLICE</td>
<td>T_HEADER</td>
<td>1536</td>
<td></td>
</tr>
</tbody>
</table>

3 rows selected

To examine the structure of a specific table, use the DESCRIBE command:
splice> describe T_DETAIL;
<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>TYPE_NAME</th>
<th>DEC</th>
<th>NUM</th>
<th>COLUMN</th>
<th>COLUMN_DEF</th>
<th>CHAR_OCTET</th>
<th>IS_NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION_HEADER_KEY</td>
<td>BIGINT</td>
<td>0</td>
<td>10</td>
<td>19</td>
<td>NULL</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>TRANSACTION_DETAIL_KEY</td>
<td>BIGINT</td>
<td>0</td>
<td>10</td>
<td>19</td>
<td>NULL</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>CUSTOMER_MASTER_ID</td>
<td>BIGINT</td>
<td>0</td>
<td>10</td>
<td>19</td>
<td>NULL</td>
<td>NULL</td>
<td>YES</td>
</tr>
<tr>
<td>TRANSACTION_DT</td>
<td>DATE</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>ORIGINALSKU_CATEGORY_ID</td>
<td>INTEGER</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>YES</td>
</tr>
</tbody>
</table>

5 rows selected

Displaying Indexes

You can display all of the indexes in a schema:

splice> show indexes in SPLICE;
<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>COLUMN_NAME</th>
<th>ORDINAL</th>
<th>NON_UNIQUE</th>
<th>TYPE</th>
<th>ASC</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_DETAIL</td>
<td>TDIDX1</td>
<td>ORIGINALSKU_CATEGORY</td>
<td>1</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>15</td>
</tr>
<tr>
<td>T_DETAIL</td>
<td>TDIDX1</td>
<td>TRANSACTIONDT</td>
<td>2</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>15</td>
</tr>
<tr>
<td>T_DETAIL</td>
<td>TDIDX1</td>
<td>CUSTOMER_MASTER_ID</td>
<td>3</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>15</td>
</tr>
<tr>
<td>T_HEADER</td>
<td>THIDX2</td>
<td>CUSTOMER_MASTER_ID</td>
<td>1</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>16</td>
</tr>
<tr>
<td>T_HEADER</td>
<td>THIDX2</td>
<td>TRANSACTIONDT</td>
<td>2</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>16</td>
</tr>
</tbody>
</table>

5 rows selected

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

splice> show indexes FROM T_DETAIL;
<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>COLUMN_NAME</th>
<th>ORDINAL</th>
<th>NON_UNIQUE</th>
<th>TYPE</th>
<th>ASC</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_DETAIL</td>
<td>TDIDX1</td>
<td>ORIGINALSKU_CATEGORY</td>
<td>1</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>15</td>
</tr>
<tr>
<td>T_DETAIL</td>
<td>TDIDX1</td>
<td>TRANSACTIONDT</td>
<td>2</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>15</td>
</tr>
<tr>
<td>T_DETAIL</td>
<td>TDIDX1</td>
<td>CUSTOMER_MASTER_ID</td>
<td>3</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>15</td>
</tr>
</tbody>
</table>

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
-------------------------------
   SPLICE          | 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:
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
-------------------------------
MYSCHEMA1 MYSCHEMA2 NULL ID
SPLICE
SQLJ
SYS
SYSCAT
SYSCS_DIAG
SYSCS_UTIL
SYSFUN
SYSLIB
SYSPROC
SYSSSTAT
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
--------------------------------------------------------
MYSCHEMA1 | MYTABLE | 1616 | |
MYSCHEMA1 | PLAYERS | 1632 | |
2 rows selected
splice> describe Players;
COLUMN_NAME | TYPE_NAME | DEC& | NUM& | COLUMN | COLUMN_DEF | CHAR_OCTET& | 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
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_SCHEMA
---------------------
MYSCHEMA1
MYSCHEMA2
NULLID
SPICE
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;
```

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:
Inserting Data

tsrule> INSERT INTO Players
1 row inserted/updated/deleted

tsrule> INSERT INTO Players
(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.
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 interpreter.
- 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.
- Our 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.

sqlshell.sh Command Line Options

You can optionally include parameter values when running sqlshell.sh script, to change default values. Here's the syntax:

```bash
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-U url</td>
<td>The full JDBC URL for connecting to your Splice Machine database.</td>
<td>-U 'jdbc:splice://xyz:1527/splicedb'</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>-h hostname</code></td>
<td>The hostname or IP address of the Splice Machine HBase Region Server.</td>
<td><code>-h splicetrial-mycluster.splicemachine.io</code></td>
</tr>
<tr>
<td><code>-p port</code></td>
<td>The port on which Splice Machine is listening. The default value is 1527.</td>
<td><code>-p 10001</code></td>
</tr>
<tr>
<td><code>-u user</code></td>
<td>Your Splice Machine database username.</td>
<td><code>-u myName</code></td>
</tr>
<tr>
<td><code>-s pass</code></td>
<td>Your Splice Machine database password.</td>
<td><code>-s myPswd</code></td>
</tr>
<tr>
<td><code>-P</code></td>
<td>Tells Splice Machine to Prompt for your password.</td>
<td><code>-P</code></td>
</tr>
<tr>
<td><code>-S</code></td>
<td>Use basic connection Security (ssl=basic) for connecting to your database.</td>
<td><code>-S</code></td>
</tr>
<tr>
<td><code>-k principal</code></td>
<td>Your kerberos principal.</td>
<td><code>-k splice</code></td>
</tr>
<tr>
<td><code>-K keytab</code></td>
<td>Your Kerberos keytab.</td>
<td><code>-K splice.keytab</code></td>
</tr>
<tr>
<td><code>-w width</code></td>
<td>The width of output rows in your window. The default width is 128.</td>
<td><code>-w 200</code></td>
</tr>
</tbody>
</table>

**NOTE:** Your keystrokes are obscured when entering your password.

**NOTE:** You must use this option when using sqlshell.sh with our Database-as-Service product.

**NOTE:** You must also specify the `-k` option when specifying this option.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-f scriptFile</td>
<td>The fully-qualified name of the SQL file to be executed.</td>
<td>-f mySqlScript.sql</td>
</tr>
<tr>
<td></td>
<td>For more information about running SQL scripts, please see our Scripting Splice Commands] tutorial.</td>
<td></td>
</tr>
<tr>
<td>-o outputFile</td>
<td>Redirects the output of a script</td>
<td>-o /tmp/myscript.out</td>
</tr>
<tr>
<td></td>
<td>This is typically used in conjunction with running a script with the -f option.</td>
<td></td>
</tr>
<tr>
<td>-q</td>
<td>Starts sqlshell in quiet mode, which suppresses the series of messages that displays when you first start sqlshell.sh.</td>
<td>-q</td>
</tr>
<tr>
<td></td>
<td>This is useful when running a script with the -f option.</td>
<td></td>
</tr>
</tbody>
</table>

**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
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:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Case Sensitive?</th>
<th>Notes and Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL keywords</td>
<td>Not case sensitive</td>
<td>These are all equivalent: SELECT, Select, select, SeLeCt.</td>
</tr>
<tr>
<td>ANSI SQL identifiers</td>
<td>Not case sensitive</td>
<td>These are not case sensitive unless they are delimited.</td>
</tr>
<tr>
<td>Java-style identifiers</td>
<td>Always case sensitive</td>
<td>These are NOT equivalent: my_name, My_Name.</td>
</tr>
</tbody>
</table>

Special Characters You Can Use
The following table describes the special characters you can use in commands:

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Character(s) to use</th>
<th>Notes and example</th>
</tr>
</thead>
<tbody>
<tr>
<td>To delimit special identifiers</td>
<td>Double quotation marks (&quot; )</td>
<td>Special identifiers are also known as delimited identifiers.</td>
</tr>
<tr>
<td>To delimit character strings</td>
<td>Single quotation marks ( ' )</td>
<td></td>
</tr>
</tbody>
</table>
### Purpose

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Character(s) to use</th>
<th>Notes and example</th>
</tr>
</thead>
<tbody>
<tr>
<td>To escape a single quote or apostrophe within a character string</td>
<td>Single quotation mark (‘)</td>
<td>Since single quotation marks are used to delimit strings, you must escape any single quotation marks you want included in the string itself.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: 'This string includes &quot;my quoted string&quot; within it.'</td>
</tr>
<tr>
<td>To escape a double quote</td>
<td>Not needed</td>
<td>You can simply include a double quotation mark in your command lines.</td>
</tr>
<tr>
<td>To specify a wild card within a Select expression</td>
<td>The asterisk (*) character</td>
<td>This is the SQL metasymbol for selecting all matching entries.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: SELECT * FROM MyTable;</td>
</tr>
<tr>
<td>To specify a wild card sequence in a string with the LIKE operator</td>
<td>The percentage (%) character</td>
<td>Example: SELECT * FROM MyTable WHERE Name LIKE 'Ga%';</td>
</tr>
<tr>
<td>To specify a single wild card character in a string with the LIKE operator</td>
<td>The underline (_) character</td>
<td>Example: SELECT * FROM MyTable WHERE Name LIKE '%Er_n%';</td>
</tr>
<tr>
<td>To begin a single-line comment</td>
<td>Two dashes (--)</td>
<td>-- the following selects everything in my table: SELECT * FROM MyTable;</td>
</tr>
<tr>
<td>To bracket a multi-line comment</td>
<td>/* and */</td>
<td>All text between the comment start /* and the comment end */ is ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/* the following selects everything in my table, which we'll then display on the screen */ SELECT * FROM MyTable;</td>
</tr>
</tbody>
</table>

### 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:

```sql
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:

```sql
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:

```sql
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:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Command Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display a list of all tables and their schemas</td>
<td><code>splice&gt; show tables;</code></td>
</tr>
<tr>
<td>Display the columns and attributes of a table</td>
<td><code>splice&gt; describe tableA;</code></td>
</tr>
<tr>
<td>Limit the number of rows returned from a select statement</td>
<td><code>splice&gt; select * from tableA { limit 10 };</code></td>
</tr>
<tr>
<td>Operation</td>
<td>Command Example</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Print a current time stamp</td>
<td><code>splice&gt; values current_timestamp;</code></td>
</tr>
</tbody>
</table>

**NOTE:** Remember that you must end your command lines with the semicolon (;) character, which submits the command line to the interpreter.
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:
<table>
<thead>
<tr>
<th><strong>Value</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaName</td>
<td>The name of the schema.</td>
</tr>
<tr>
<td>tableName</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>partition</td>
<td>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.</td>
</tr>
<tr>
<td>rowsCollected</td>
<td>The total number of rows collected for the table.</td>
</tr>
<tr>
<td>partitionSize</td>
<td>The combined size of the table's partitions.</td>
</tr>
<tr>
<td>statsType</td>
<td>The type of statistics, which is one of these values:</td>
</tr>
<tr>
<td></td>
<td>0 Full table (not sampled) statistics that reflect the unmerged partition values.</td>
</tr>
<tr>
<td></td>
<td>1 Sampled statistics that reflect the unmerged partition values.</td>
</tr>
<tr>
<td></td>
<td>2 Full table (not sampled) statistics that reflect the table values after all partitions have been merged.</td>
</tr>
<tr>
<td></td>
<td>3 Sampled statistics that reflect the table values after all partitions have been merged.</td>
</tr>
<tr>
<td>sampleFraction</td>
<td>The sampling percentage, expressed as 0.0 to 1.0,</td>
</tr>
<tr>
<td></td>
<td>If statsType=0 or statsType=1 (full statistics), this value is not used, and is shown as 0.</td>
</tr>
<tr>
<td></td>
<td>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).</td>
</tr>
</tbody>
</table>

**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 in the schema.

**NOTE:** This command operates like the SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS built-in system procedure.

### Examples

```sql
splice> analyze table test.t2;
```

```
+----------------+-----+----------+----------+----------------+-------------+------------------+
<table>
<thead>
<tr>
<th>schemaName</th>
<th>tableName</th>
<th>partition</th>
<th>rowsCollected</th>
<th>partitionSize</th>
<th>partitionCount</th>
<th>statsType</th>
<th>sampleFraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST</td>
<td>T2</td>
<td>-All-</td>
<td>39226</td>
<td>235356</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1 rows selected
``` 

```sql
splice> analyze table test.t2 estimate statistics sample 50 percent;
```

```
+----------------+-----+----------+----------+----------------+-------------+------------------+
<table>
<thead>
<tr>
<th>schemaName</th>
<th>tableName</th>
<th>partition</th>
<th>rowsCollected</th>
<th>partitionSize</th>
<th>partitionCount</th>
<th>statsType</th>
<th>sampleFraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST</td>
<td>T2</td>
<td>-All-</td>
<td>19613</td>
<td>235356</td>
<td>1</td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1 rows selected
``` 

```sql
splice> analyze schema test;
```

```
+----------------+-----+----------+----------+----------------+-------------+------------------+
<table>
<thead>
<tr>
<th>schemaName</th>
<th>tableName</th>
<th>partition</th>
<th>rowsCollected</th>
<th>partitionSize</th>
<th>partitionCount</th>
<th>statsType</th>
<th>sampleFraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST</td>
<td>T2</td>
<td>-All-</td>
<td>39226</td>
<td>235356</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1 rows selected
``` 

```sql
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

```plaintext
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/splicedb';
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/splicedb
CONNECTION1* - jdbc:splice://xyz:1527/splicedb
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) statement 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.

compression
A Boolean or string value that specifies how to compress the exported files; you can specify one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The exported files are compressed using gzip.</td>
</tr>
<tr>
<td>false</td>
<td>Exported files are not compressed.</td>
</tr>
<tr>
<td>'gz' or 'gzip'</td>
<td>The exported files are compressed using gzip.</td>
</tr>
<tr>
<td>'bz2' or 'bzip2'</td>
<td>The exported files are compressed using bzip2.</td>
</tr>
<tr>
<td>'none'</td>
<td>Exported files are not compressed.</td>
</tr>
</tbody>
</table>

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.


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.a2;

-- This example explicitly specifies options:
splice> EXPORT('/my/export/dir', 'bz2', 3, 'utf-8', '|', ';')
   SELECT a,b,sqrt(c) FROM t1 join t2 on t1.a=t2.a2;
See Also

- The Export_Binary command exports query results in binary format.
Export_Binary Command

The `export_binary` command exports the results of an SQL query to one or more binary files.

This command is currently limited to writing binary files only in parquet format; other formats will be supported in a future release.

Syntax

```
EXPORT_BINARY ( exportPath,
    compression,
    format ) <SQL_QUERY>;
```

- **exportPath**: The directory in which you want the export file(s) written.

- **compression**: A Boolean or string value that specifies how to compress the exported files; you can specify one of the following values:
  
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The exported files are compressed using snappy.</td>
</tr>
<tr>
<td>false</td>
<td>Exported files are not compressed.</td>
</tr>
<tr>
<td>'snappy'</td>
<td>The exported files are compressed using snappy.</td>
</tr>
<tr>
<td>'none'</td>
<td>Exported files are not compressed.</td>
</tr>
</tbody>
</table>

- **format**: The format in which to write the exported file(s). The only format supported at this time is parquet.

Usage

The `EXPORT_BINARY` command generates one or more binary files and stores them in the directory that you specify in the `exportPath` parameter. More than one output file can be generated to enhance the parallelism and performance of this operation.

If `compression=true`, then each of the generated files is named with this format:
If \( \text{compression=false} \), then each of the generated files is named with this format:

\[
\text{part-r-<N>.parquet}
\]

The value of \( <N> \) is a sequence of numbers and letters.

**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 files to your `localPath`.


**Examples**

```
splice> EXPORT_BINARY('/my/export/dir', true, 'parquet')
    SELECT a,b,sqrt(c) FROM t1 WHERE a > 100;
```

**See Also**

- The Export command exports query results in CSV format.
**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;
<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
<tr>
<td>dessert</td>
<td>creme brulee</td>
<td>7</td>
</tr>
<tr>
<td>appetizer</td>
<td>baby greens</td>
<td>7</td>
</tr>
<tr>
<td>dessert</td>
<td>rhubarb pie</td>
<td>4</td>
</tr>
</tbody>
</table>
4 rows selected
splice> rollback;
splice> SELECT * FROM menu;
<table>
<thead>
<tr>
<th>COURSE</th>
<th>ITEM</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>entree</td>
<td>lamb chop</td>
<td>14</td>
</tr>
<tr>
<td>dessert</td>
<td>creme brulee</td>
<td>7</td>
</tr>
<tr>
<td>appetizer</td>
<td>baby greens</td>
<td>7</td>
</tr>
</tbody>
</table>
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:

```sql
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/splicedb;user=YourUserId;password=YourPassword';

And this example includes specifies that SSL peer authentication will be enabled for the connection:

splice> connect 'jdbc:splice://1.2.3.456:1527/splicedb;user=YourUserId;password=YourPassword;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.

```plaintext
splice> connect 'jdbc:splice://localhost:1527/splicedb';
```

Here is an example that includes a user ID and password in the connect string:

```plaintext
splice> connect 'jdbc:splice://localhost:1527/splicedb;user=joey;password=bossman';
```
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:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN_NAME</td>
<td>The name of the column</td>
</tr>
<tr>
<td>TYPE_NAME</td>
<td>The data type of the column</td>
</tr>
<tr>
<td>DECIMAL_DIGITS</td>
<td>The number of fractional digits</td>
</tr>
<tr>
<td>NUM_PREC_RADIX</td>
<td>The radix, which is typically either 10 or 2</td>
</tr>
<tr>
<td>COLUMN_SIZE</td>
<td>The column size:</td>
</tr>
<tr>
<td></td>
<td>▶ For char or date types, this is the maximum number of characters</td>
</tr>
<tr>
<td></td>
<td>▶ For numeric or decimal types, this is the precision</td>
</tr>
<tr>
<td>COLUMN_DEF</td>
<td>The default value for the column</td>
</tr>
<tr>
<td>CHAR_OCTET</td>
<td>Maximum number of bytes in the column</td>
</tr>
<tr>
<td>IS_NULL</td>
<td>Whether (YES) or not (NO) the column can contain null values</td>
</tr>
</tbody>
</table>
Examples

splice> describe T_DETAIL;

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>TYPE_NAME</th>
<th>DEC</th>
<th>NUM</th>
<th>COLUM</th>
<th>COLUMN_DEF</th>
<th>CHAR_OCTET</th>
<th>IS_NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION_HEADER_KEY</td>
<td>BIGINT</td>
<td>0</td>
<td>10</td>
<td>19</td>
<td>NULL</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>TRANSACTION_DETAIL_KEY</td>
<td>BIGINT</td>
<td>0</td>
<td>10</td>
<td>19</td>
<td>NULL</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>CUSTOMER_MASTER_ID</td>
<td>BIGINT</td>
<td>0</td>
<td>10</td>
<td>19</td>
<td>NULL</td>
<td>NULL</td>
<td>YES</td>
</tr>
<tr>
<td>TRANSACTION_DT</td>
<td>DATE</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>ORIGINALSKU_CATEGORY_ID</td>
<td>INTEGER</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>YES</td>
</tr>
</tbody>
</table>

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**

```plaintext
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 syno
nyms
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
splice> -- 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=YourUserId;password=YourPassword' 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/splicedb
* = current connection
splice(SAMPLE2)> set connection sample1;
splice(SAMPLE1)> disconnect all;
splice>
```
The **SET SESSION_PROPERTY** command sets or unsets specific session-level property values. Session-level properties assign an initial value to certain Splice Machine query hints, which allows you to supply the hint by default in all queries.

This command should **only be used by advanced users**. Query hints are very powerful, and applying them by default can lead to unforeseen and unfortunate consequences.

**Syntax**

```
SET SESSION_PROPERTY {
    propertySetting [, propertySetting]*
}
```

`propertySetting`

- `useSpark = [true | false | null]`

**NOTE:** Splice Machine will include additional session properties in the future.

**Usage**

Specifying a hint in a query overrides the session-level property setting. For example, if you use this command to tell Splice Machine to use Spark to process every query in the session:

```
SET SESSION_PROPERTY useSpark=true
```

and then submit a query with the hint:

```
--splice-properties useSpark=false
```

That query will run on the control side (using HBase), and not on Spark. Other, unhinted queries will continue to assume the `useSpark=true` default hint value.

The following table summarizes the currently available session properties:
### Session Property Description

<table>
<thead>
<tr>
<th>Session Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| useSpark         | Setting this to **true** is equivalent to having the useSpark=true hint applied by default for every query.  
Setting this to **false** is equivalent to specifying the useSpark=false hint applied by default for every query.  
Setting this to **null** unsets the property, which is equivalent to not specifying the useSpark=false hint by default in any query. |

#### Unsetting a Property

To unset a session property, assign null as its value.

#### Displaying Current Session Properties

You can use the values current session_property expression to display the current session_property values. Note that only properties that you have explicitly set to non-null values are displayed.

#### Examples

Here's an example of setting a session property:

```
splice> set session_property useSpark=true;
0 rows inserted/updated/deleted
splice> values current session_property;
1
------------------------------------------------------------------
USESPARK=true;
1 row selected
```

And here's an example of unsetting those values (resetting them to their connection default values):

```
splice> set session_property useSpark=null;
0 rows inserted/updated/deleted
splice> values current session_property;
1
------------------------------------------------------------------
1 row selected
```
See Also

» Query Optimization
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/splicedb
* = current connection
splice(NEWDB)>
```
Show Create Table

The `show create table` command displays the DDL used with the `create table` statement to create a specified table.

**Syntax**

```
SHOW CREATE TABLE schemaName.tableName
```

*`schemaName`*

The schema name. If not specified, the current schema is used.

*`tableName`*

The table name.

**Example**

```sql
splice> CREATE TABLE Players(
    ID           SMALLINT NOT NULL,
    Team         VARCHAR(64) NOT NULL,
    Name         VARCHAR(64) NOT NULL,
    Position     CHAR(2),
    DisplayName  VARCHAR(24),
    BirthDate    DATE
);
0 rows inserted/updated/deleted

show create table players;
DDL
---
CREATE TABLE "SPLICE"."PLAYERS" (
"ID" SMALLINT NOT NULL
,"TEAM" VARCHAR(64) NOT NULL
,"NAME" VARCHAR(64) NOT NULL
,"POSITION" CHAR(2)
,"DISPLAYNAME" VARCHAR(24)
,"BIRTHDATE" DATE
) ;
1 row selected
```

**See Also**

- The `SYSCS_UTIL.SHOW_CREATE_TABLE` built-in system procedure.
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:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION_SCHEMA</td>
<td>The name of the function's schema</td>
</tr>
<tr>
<td>FUNCTION_NAME</td>
<td>The name of the function</td>
</tr>
<tr>
<td>REMARKS</td>
<td>Any remarks that have been stored for the function</td>
</tr>
</tbody>
</table>

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;
 splice> show functions in splice;
FUNCTION_SCHEMA|FUNCTION_NAME               |REMARKS
-------------------------------------------------------------------------
SPLICE        |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:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_NAME</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>INDEX_NAME</td>
<td>The name of the index.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>The name of the column.</td>
</tr>
<tr>
<td>ORDINAL</td>
<td>The position of the column in the index.</td>
</tr>
<tr>
<td>NON_UNIQUE</td>
<td>Whether this is a unique or non-unique index.</td>
</tr>
<tr>
<td>TYPE</td>
<td>The index type</td>
</tr>
<tr>
<td>ASC_</td>
<td>Indicates if this is an ascending (A) or descending (D) index.</td>
</tr>
<tr>
<td>CONGLOM_NO</td>
<td>The conglomerate number, which points to the corresponding table in HBase.</td>
</tr>
</tbody>
</table>
### Examples

splice> show indexes from my_table;

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>INDEX_NAME</th>
<th>COLUMN_NAME</th>
<th>ORDINAL</th>
<th>NON_UNIQUE</th>
<th>TYPE</th>
<th>ASC</th>
<th>CONGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MY_TABLE</td>
<td>I1</td>
<td>ID</td>
<td>1</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>1937</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I1</td>
<td>STATE_CD</td>
<td>2</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>1937</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I1</td>
<td>CITY</td>
<td>3</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>1937</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I2</td>
<td>ID</td>
<td>1</td>
<td>true</td>
<td>BTREE</td>
<td>D</td>
<td>1953</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I2</td>
<td>STATE_CD</td>
<td>2</td>
<td>true</td>
<td>BTREE</td>
<td>D</td>
<td>1953</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I2</td>
<td>CITY</td>
<td>3</td>
<td>true</td>
<td>BTREE</td>
<td>D</td>
<td>1953</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I3</td>
<td>ID</td>
<td>1</td>
<td>true</td>
<td>BTREE</td>
<td>D</td>
<td>1969</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I3</td>
<td>STATE_CD</td>
<td>2</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>1969</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I3</td>
<td>CITY</td>
<td>3</td>
<td>true</td>
<td>BTREE</td>
<td>D</td>
<td>1969</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I4</td>
<td>LATITUDE</td>
<td>1</td>
<td>true</td>
<td>BTREE</td>
<td>D</td>
<td>1985</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I4</td>
<td>STATE_CD</td>
<td>2</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>1985</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I4</td>
<td>ID</td>
<td>3</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>1985</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I5</td>
<td>ID</td>
<td>1</td>
<td>true</td>
<td>BTREE</td>
<td>A</td>
<td>2001</td>
</tr>
<tr>
<td>MY_TABLE</td>
<td>I5</td>
<td>ID</td>
<td>2</td>
<td>true</td>
<td>BTREE</td>
<td>D</td>
<td>2001</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_NAME</td>
<td>The name of the table</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>The name of the column</td>
</tr>
<tr>
<td>KEY_SEQ</td>
<td>The order of the column within the primary key</td>
</tr>
<tr>
<td>PK_NAME</td>
<td>The unique name of the constraint</td>
</tr>
</tbody>
</table>

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

```sql
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:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCEDURE_SCHEMA</td>
<td>The name of the procedure's schema</td>
</tr>
<tr>
<td>PROCEDURE_NAME</td>
<td>The name of the procedure</td>
</tr>
<tr>
<td>REMARKS</td>
<td>Any remarks that have been stored for the procedure</td>
</tr>
</tbody>
</table>
Examples
splice> show procedures in syscs_util;

<table>
<thead>
<tr>
<th>PROCEDURE_SCHEMA</th>
<th>PROCEDURE_NAME</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYCS_UTIL</td>
<td>BULK_IMPORT_HFILE</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>COLLECT_NONMERGED_TABLE_SAMPLE_STATISTICS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>COLLECT_NONMERGED_TABLE_STATISTICS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>COLLECT_SCHEMA_STATISTICS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>COLLECT_TABLE_SAMPLE_STATISTICS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>COLLECT_TABLE_STATISTICS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>COMPACT_REGION</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>COMPUTE_SPLIT_KEY</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>DELETE_REGION</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>DELETE_SNAPSHOT</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>DISABLE_ALL_COLUMN_STATISTICS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>DISABLE_COLUMN_STATISTICS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>DROP_SCHEMA_STATISTICS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>DROP_TABLE_STATISTICS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>ENABLE_ALL_COLUMN_STATISTICS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>ENABLE_COLUMN_STATISTICS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>GET_ACTIVATION</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>GET_ENCODED_REGION_NAME</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>GET_REGIONS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>GET_START_KEY</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>IMPORT_DATA</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>IMPORT_DATA_UNSAFE</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>INVALIDATE_DICTIONARY_CACHE</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>INVALIDATE_GLOBAL_DICTIONARY_CACHE</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>MAJOR_COMPACT_REGION</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>MERGE_DATA_FROM_FILE</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>MERGE_REGIONS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>RESTORE_SNAPSHOT</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SAMPLE_DATA</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SET_PURGE_DELETED_ROWS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SNAPSHOT_SCHEMA</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>MERGE_REGIONS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SNAPSHOT_TABLE</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_BACKUP_DATABASE</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_BACKUP_DATABASE_ASYNC</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_CREATE_USER</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_DELETE_BACKUP</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_DELETE_OLD_BACKUPS</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_Dictionary_DELETE</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_DROP_USER</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_ELEVATE_TRANSACTION</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_EMPTY_GLOBAL_STATEMENT_CACHE</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_EMPTY_STATEMENT_CACHE</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_ENABLE_ENTERPRISE</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_FLUSH_TABLE</td>
<td>com.splicemachine.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Class</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_ACTIVE_SERVERS</td>
<td>retrieves the active servers in the system.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_ACTIVE_TRANSACTION_IDS</td>
<td>retrieves the active transaction IDs in the system.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_ALL_PROPERTIES</td>
<td>retrieves all properties of the system.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_AUTO_INCREMENT_ROW_LOCATIONS</td>
<td>retrieves the auto-increment row locations.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_CACHE_INFO</td>
<td>retrieves cache information.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_CURRENT_TRANSACTION</td>
<td>retrieves the current transaction in the system.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_EXEC_SERVICE_INFO</td>
<td>retrieves the exec service information.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_GLOBAL_DATABASE_PROPERTY</td>
<td>retrieves the global database property.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_LOGGERS</td>
<td>retrieves the loggers in the system.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_LOGGER_LEVEL</td>
<td>retrieves the logger level.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_REGION_SERVER_CONFIG_INFO</td>
<td>retrieves the region server configuration information.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_REGION_SERVER_STATS_INFO</td>
<td>retrieves the region server statistics information.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_REQUESTS</td>
<td>retrieves the requests in the system.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_RUNNING_OPERATIONS</td>
<td>retrieves the running operations in the system.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_RUNNING_OPERATIONS_LOCAL</td>
<td>retrieves the running operations local.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_SCHEMA_INFO</td>
<td>retrieves the schema information.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_SESSION_INFO</td>
<td>retrieves the session information.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_SPICE_TOKEN</td>
<td>retrieves the spice token.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_STORED_STATEMENT_PLAN_INFO</td>
<td>retrieves the stored statement plan information.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_TIMESTAMP_GENERATOR_INFO</td>
<td>retrieves the timestamp generator information.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_TIMESTAMP_REQUEST_INFO</td>
<td>retrieves the timestamp request information.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_TOTAL_CACHE_INFO</td>
<td>retrieves the total cache information.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_VERSION_INFO</td>
<td>retrieves the version information.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_GET_WRITE_INTAKE_INFO</td>
<td>retrieves the write intake information.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_HBASE_OPERATION</td>
<td>performs a HBase operation.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_HDFS_OPERATION</td>
<td>performs an HDFS operation.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_INVALIDATE_STORED_STATEMENTS</td>
<td>invalidates the stored statements.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_KILL_OPERATION</td>
<td>kills an operation.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_KILL_OPERATION_LOCAL</td>
<td>kills an operation local.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_KILL_STALE_TRANSACTIONS</td>
<td>kills stale transactions.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_KILL_TRANSACTION</td>
<td>kills a transaction.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_MODIFY_PASSWORD</td>
<td>modifies the password.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA</td>
<td>performs a major compaction on the schema.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_PERFORM_MAJOR_COMPACTION_ON_TABLE</td>
<td>performs a major compaction on the table.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_RECOMPILE_INVALID_STORED_STATEMENTS</td>
<td>recompiles invalid stored statements.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_REFRESH_EXTERNAL_TABLE</td>
<td>refreshes an external table.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_RELOAD_SECURITY_POLICY</td>
<td>reloads the security policy.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_RESET_PASSWORD</td>
<td>resets the password.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_RESTORE_DATABASE</td>
<td>restores the database.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_RESTORE_DATABASE_ASYNC</td>
<td>restores the database async.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_RESTORE_DATABASE_OWNER</td>
<td>restores the database owner.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_SET_DATABASE_PROPERTY</td>
<td>sets the database property.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_SET_GLOBAL_DATABASE_PROPERTY</td>
<td>sets the global database property.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_SET_LOGGER_LEVEL</td>
<td>sets the logger level.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_SPLIT_REGION_AT_POINTS</td>
<td>splits a region at points.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_SPLIT_TABLE</td>
<td>splits a table.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_SPLIT_TABLE_AT_POINTS</td>
<td>splits a table at points.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_SPLIT_TABLE_EVENLY</td>
<td>splits a table evenly.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_SPLIT_TABLE_OR_INDEX</td>
<td>splits a table or index.</td>
</tr>
<tr>
<td>SYSCS_UTIL</td>
<td>SYCS_SPLIT_TABLE_OR_INDEX_AT_POINTS</td>
<td>splits a table or index at points.</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_START_CHILD_TRANSACTION</td>
<td>com.splicemachine. ...</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_UPDATE_ALL_SYSTEM_PROCEDURES</td>
<td>com.splicemachine. ...</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_UPDATE_METADATA_STORED_STATEMENTS</td>
<td>com.splicemachine. ...</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_UPDATE_SCHEMA_OWNER</td>
<td>com.splicemachine. ...</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>SYCS_UPDATE_SYSTEM_PROCEDURE</td>
<td>com.splicemachine. ...</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>UPSERT_DATA_FROM_FILE</td>
<td>com.splicemachine. ...</td>
</tr>
<tr>
<td>SYCS_UTIL</td>
<td>VACUUM</td>
<td>com.splicemachine. ...</td>
</tr>
</tbody>
</table>

107 rows selected
Show Roles

The `show roles` command a sorted list of all of the roles that have been defined in the database.

**NOTE:** Use the “`values current_role;`” statement to display the list of roles that are active in the current session.

**Syntax**

```
SHOW ROLES
```

**Examples**

```
splice> create role testRole;
0 rows inserted/updated/deleted
splice> show roles;

ROLEID
-----------------------
ADMIN, EDITOR, WRITER

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:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_SCHEMA</td>
<td>The name of the synonym's schema</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>The name of the table</td>
</tr>
<tr>
<td>CONGLOM_ID</td>
<td>The conglomerate number, which points to the corresponding table in HBase</td>
</tr>
<tr>
<td>REMARKS</td>
<td>Any remarks associated with the table</td>
</tr>
</tbody>
</table>

**Examples**

```
splice> show synonyms;
TABLE_SCHEMA | TABLE_NAME | CONGLOM_ID | REMARKS
SPLICE        | 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:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_SCHEMA</td>
<td>The name of the table's schema</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>The name of the table</td>
</tr>
<tr>
<td>CONGLOM_ID</td>
<td>The conglomerate number, which points to the corresponding table in HBase</td>
</tr>
<tr>
<td>REMARKS</td>
<td>Any remarks associated with the table</td>
</tr>
</tbody>
</table>
### Examples

splice> show tables;

<table>
<thead>
<tr>
<th>TABLE_SCHEM</th>
<th>TABLE_NAME</th>
<th>CONGLOM_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>SYSALIASES</td>
<td>352</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSBACKUP</td>
<td>1040</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSBACKUPITEMS</td>
<td>1136</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCHECKS</td>
<td>368</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCOLPERMS</td>
<td>704</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCOLUMNS</td>
<td>64</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCOLUMNSTATS</td>
<td>1216</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCONGLOMERATES</td>
<td>80</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSCONSTRAINTS</td>
<td>320</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSDEPENDS</td>
<td>240</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSFILES</td>
<td>256</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSFOREIGNKEYS</td>
<td>288</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSKEYS</td>
<td>272</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSPERMS</td>
<td>784</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSPHYSICALSTATS</td>
<td>1232</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSPRIMARYKEYS</td>
<td>384</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSROLES</td>
<td>736</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSRoutinePERMS</td>
<td>720</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSCHEMAPERMS</td>
<td>1392</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSCHEMAs</td>
<td>32</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSequENCES</td>
<td>752</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSSTATEMENTS</td>
<td>304</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTablePERMS</td>
<td>688</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTables</td>
<td>48</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTableSTATS</td>
<td>1248</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTRiggers</td>
<td>656</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSUsers</td>
<td>816</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSVIEWS</td>
<td>336</td>
</tr>
<tr>
<td>SYSIBM</td>
<td>SYSDUMMY1</td>
<td>1344</td>
</tr>
<tr>
<td>SPLICE</td>
<td>MYTABLE</td>
<td>1536</td>
</tr>
</tbody>
</table>

32 rows selected

splice>show tables in SPLICE;

<table>
<thead>
<tr>
<th>TABLE_SCHEM</th>
<th>TABLE_NAME</th>
<th>CONGLOM_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPLICE</td>
<td>MYTABLE</td>
<td>1536</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_SCHEMA</td>
<td>The name of the table's schema</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>The name of the table</td>
</tr>
<tr>
<td>CONGLOM_ID</td>
<td>The conglomerate number, which points to the corresponding table in HBase</td>
</tr>
<tr>
<td>REMARKS</td>
<td>Any remarks associated with the table</td>
</tr>
</tbody>
</table>

**Examples**

```
splice> show views;
TABLE_SCHEMA | TABLE_NAME   | CONGLOM_ID | REMARKS
SPLICE        | GUITAR_BRANDS| 4321        |
0 rows selected
```
Scripting the splice> Command Line Interface

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.

   ```sql
   connect 'jdbc:splice://localhost:1527/splicedb;user=YourUserId;password=YourPassword';
   
   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    
               (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=YourUserId;password=YourPassword';
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  
    (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;

<table>
<thead>
<tr>
<th>ID</th>
<th>TEAM</th>
<th>NAME</th>
<th>POS&amp;</th>
<th>DISPLAYNAME</th>
<th>BIRTHDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>Cards</td>
<td>Earl Hastings</td>
<td>OF</td>
<td>Speedy Earl</td>
<td>1982-04-22</td>
</tr>
<tr>
<td>73</td>
<td>Giants</td>
<td>Lester Johns</td>
<td>P</td>
<td>Big John</td>
<td>1984-06-09</td>
</tr>
<tr>
<td>99</td>
<td>Giants</td>
<td>Joe Bojangles</td>
<td>C</td>
<td>Little Joey</td>
<td>1991-07-11</td>
</tr>
</tbody>
</table>

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:

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

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
           (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:

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

3. **Run the script**

In your terminal window, invoke the script:

```bash
./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=YourUserId;password=YourPassword';
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
   (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;

+-------+--------+-------------------+---+---------------------------+----------+
<table>
<thead>
<tr>
<th></th>
<th>TEAM</th>
<th>NAME</th>
<th>POS</th>
<th>DISPLAYNAME</th>
<th>BIRTHDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>Cards</td>
<td>Earl Hastings</td>
<td>OF</td>
<td>Speedy Earl</td>
<td>1982-04-22</td>
</tr>
<tr>
<td>73</td>
<td>Giants</td>
<td>Lester Johns</td>
<td>P</td>
<td>Big John</td>
<td>1984-06-09</td>
</tr>
<tr>
<td>99</td>
<td>Giants</td>
<td>Joe Bojangles</td>
<td>C</td>
<td>Little Joey</td>
<td>1991-07-11</td>
</tr>
</tbody>
</table>
+-------+--------+-------------------+---  +---------------------------+----------+
3 rows selected

Using nohup 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.
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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL-@</td>
<td>Set mark</td>
</tr>
<tr>
<td>CTRL-A</td>
<td>Move to the beginning of the line</td>
</tr>
<tr>
<td>CTRL-B</td>
<td>Move back one character</td>
</tr>
<tr>
<td>CTRL-D</td>
<td>Delete the highlighted character</td>
</tr>
<tr>
<td>CTRL-E</td>
<td>Move to the end of the line</td>
</tr>
<tr>
<td>CTRL-F</td>
<td>Move forward one character</td>
</tr>
<tr>
<td>CTRL-H</td>
<td>Backward delete character</td>
</tr>
<tr>
<td>CTRL-J</td>
<td>Accept (submit) the line</td>
</tr>
<tr>
<td>CTRL-L</td>
<td>Clear the screen</td>
</tr>
<tr>
<td>CTRL-M</td>
<td>Accept the line</td>
</tr>
<tr>
<td>CTRL-N</td>
<td>Move to the next line in history</td>
</tr>
<tr>
<td>CTRL-P</td>
<td>Move to the previous line in history</td>
</tr>
<tr>
<td>CTRL-R</td>
<td>Reverse search through your command line history</td>
</tr>
<tr>
<td>CTRL-S</td>
<td>Forward search through your command line history</td>
</tr>
<tr>
<td>CTRL-T</td>
<td>Transpose characters: switch the highlighted character with the one preceding it</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>CTRL-U</td>
<td>Discard from the cursor position to the beginning of the line</td>
</tr>
<tr>
<td>CTRL-]</td>
<td>Search for a character on the line</td>
</tr>
<tr>
<td>CTRL-__</td>
<td>Undo</td>
</tr>
<tr>
<td>ALT-&lt;</td>
<td>Go to the beginning of the history</td>
</tr>
<tr>
<td>ALT--&gt;</td>
<td>Go to the end of the history</td>
</tr>
<tr>
<td>ALT-B</td>
<td>Backward word</td>
</tr>
<tr>
<td>ALT-C</td>
<td>Capitalize the current word</td>
</tr>
<tr>
<td>ALT-F</td>
<td>Forward word</td>
</tr>
<tr>
<td>ALT-L</td>
<td>Downcase word</td>
</tr>
<tr>
<td>ALT-R</td>
<td>Revert line</td>
</tr>
<tr>
<td>ALT-T</td>
<td>Transpose words</td>
</tr>
<tr>
<td>ALT-U</td>
<td>Uppercase word</td>
</tr>
</tbody>
</table>

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.
**Splice Machine ML Manager**

The Splice ML Manager is an integrated machine learning (ML) platform that minimizes data movement and enables enterprises to deliver better decisions faster by continuously training the models on the most updated available data. With Splice ML Manager, data science teams are able to produce a higher number of more predictive models, facilitated by the ability to:

- Experiment frequently using diverse parameters to compare model effectiveness
- Leverage updated operational data to concurrently train the model
- Minimize the movement of data by running the models on your cluster's Spark executors
- Compress the time from model deployment to action

Splice ML Manager provides end-to-end life-cycle management for your ML models, thereby streamlining and accelerating the design and deployment of intelligent applications using real-time data. ML Manager through its tight integration with Splice data platform results in reduced data movement that empowers data scientists to conduct a higher number of experiments to derive better feature vectors with more signal and compare algorithms with varied parameters to build better models in a limited amount of time.

**Overview**

The *Splice ML Manager* facilitates machine learning development within Zeppelin notebooks. Here are some of its key features:

- *ML Manager* runs directly on Apache Spark, allowing you to complete massive jobs in parallel.
- Our native `PySpliceContext` lets you directly access the data in your database and very efficiently convert it to/from a Spark DataFrame with no serialization/deserialization required.
- MLflow is integrated directly into your Splice Machine cluster, to facilitate tracking of your entire Machine Learning workflow.
- After you have found the best model for your task, you can easily deploy it live to AWS SageMaker to make predictions in real time.
- As new data flows in, updating your model is a simple matter of returning to your Notebook, creating new runs, and redeploying by tapping a button.

The Splice ML Manager leverages Apache MLflow and Amazon Sagemaker, and like MLflow, is organized around the concepts of *runs*:

A run is the execution of some data science code; each run can record different types of information, including:

<table>
<thead>
<tr>
<th>Information Type</th>
<th>Purpose</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Metrics</em></td>
<td>Map string values such as F1 to double-precision numbers such as 0.85.</td>
<td>Model output metrics, such as: <em>F1 score, AUC, Precision, Recall, R^2</em>.</td>
</tr>
<tr>
<td>Information Type</td>
<td>Purpose</td>
<td>Examples</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>Parameters</td>
<td>Map strings such as <code>classifier</code> to strings such as <code>DecisionTree</code>.</td>
<td>Model parameters, such as Num Trees, Preprocessing Steps, Regularization.</td>
</tr>
<tr>
<td>Models</td>
<td>So that you can subsequently deploy them to SageMaker.</td>
<td>Fitted pipelines or models.</td>
</tr>
<tr>
<td>Tags</td>
<td>These map strings such as <code>deployable</code> to strings such as <code>true</code>.</td>
<td>Specific pieces of information associated with a run, such as the project, version, and deployable status.</td>
</tr>
</tbody>
</table>

For more information about logging information in the ML Manager, see the ML Manager API topic.

ML Manager organizes runs into *experiments*; each experiment groups runs together for a specific task; for example, you might experiment with using different machine learning models in different runs, to compare results.

**Using ML Manager**

The other topics in this section will help you to start using ML Manager:

- Using the ML Manager provides an overview of how you can use MLManager to develop, tune, and deploy your Machine Learning projects.
- The ML Manager API describes the application programming interface you can use in your code to interact with Splice ML Manager.
Using the Splice ML Manager

This topic shows you how to use the Splice Machine ML Manager, a machine learning framework that combines the power of Splice Machine with the power of Apache Zeppelin notebooks, Apache MLflow, and Amazon Sagemaker to create a full-cycle platform for developing and maintaining your smart applications. This topic is organized into these sections:

- **ML Manager Workflow** provides a quick overview of what the ML Manager does and how it interfaces with MLflow and SageMaker to provide a complete Machine Learning production environment.
- **Running an Experiment** walks you through creating an ML experiment for detecting credit card fraud and shows you how to train, run, and compare two different learning models.
- **Deploying Your Model to AWS SageMaker** walks you through deploying your model on AWS.
- **Retraining the Model with New Data** shows you how to retrain your model with new data and update your deployment.

The ML Manager Introduction topic in this chapter provides an overview of the ML Manager, and the ML Manager API topic provides reference information for its API.

**ML Manager WorkFlow**

Here's what the basic flow of processes involved in developing, tuning, and deploying your ML projects looks like with ML Manager:

...
The basic workflow is:

1. Work with MLlib and other machine learning libraries in a Zeppelin notebook to directly interact with Spark and your Splice Machine database.
2. Use MLflow within your notebook to create experiments and runs, and to track variables, parameters, and other information about your runs.
3. Use the MLflow Tracking UI to monitor information about your experiments and runs.
4. Iterate on your experiments until you develop the learning model that you want to deploy, using the tracking UI to compare your runs.
5. Use the Splice ML Jobs Tracker to deploy your model to AWS SageMaker, by simply filling in a few form
fields and clicking a button.

6. Write Apps that use SageMaker's RESTful API to interface with your deployed model.

7. As new data arrives, you can return to Step 1 and repeat the process.

About MLflow

MLflow is an open source platform for managing the end-to-end machine learning lifecycle; with MLflow and Splice ML Manager, you can:

- Track your model training sessions, which are called runs.
- Group a collection of runs under an experiment, which allows you to visualize and compare a set of runs, and to download run artifacts for analysis by other tools.
- View your experiments in the MLflow Tracking UI, which you access by pointing your browser at port 5001.

About Storing Models and Pipelines

You can save your pipeline and model to S3 using the save method of MLlib Pipeline or MLlib model objects. Assuming that you've created a pipeline and built a model, you can save them as follows:

```python
%spark.pyspark
model.save('s3a://splice-demo/fraudDemoPipelineModel')
```

or

```python
%spark.pyspark
pipeline.save('s3a://splice-demo/fraudDemoPipeline')
```

If you are developing or have developed a model that you expect to deploy in the future, you should save the model to MLflow. Assuming that you've previously created an instance of MLManager named manager and have created a MLlib model named model, you can save the model to MLflow with this statement:

```python
%spark.pyspark
manager.log_spark_model(model)
```

The code in the Running an Experiment section below contains examples of saving models to S3 and to MLflow.

About SageMaker

Amazon SageMaker allows you to easily deploy the machine learning models that you develop with the Splice ML Manager on Amazon AWS. The only requirement is that you have an ECR repository set up on AWS; ECR is Amazon's fully-managed Docker container registry that simplifies deploying Docker images and is integrated with Amazon's Elastic Container Service (ECS).
When you tell our *ML Manager* to deploy a model to SageMaker, *ML Manager* creates a Docker image and uploads it to your ECR repository. You can specify which AWS instance types you want to deploy on, and how many instances you want to deploy. We send the deployment request to SageMaker, which creates an endpoint, launches your ML compute instances, and the deploys your model to them.

You can also use the same process to deploy an updated version of your model.

**Running an Experiment**
This section walks you through creating and running an experiment with ML Manager, in these steps:

- **Preparing Your Experiment**
- **The First Run**
- **Trying a Different Model**

The Splice ML Manager, along with mlFlow, allows you to group a set of *runs* into an *experiment*. Each run can use different values and parameters, all of which can be easily tracked and evaluated with help from the mlFlow user interface.

**Preparing Your Experiment**
In this section, we'll prepare our first experiment, in these steps:

1. **Connect to your database**
2. **Load the data into your database**
3. **Try visualizing the data in Zeppelin**
4. **Create your *MLManager* instance**
5. **Create a new experiment**
6. **Load the database table directly into a Spark DataFrame**
7. **View your Experiment in the Mlflow UI**

**1. Connect to your database**
First, let's establish a connection to your database using Python via our Native Spark Datasource. We will use the SpliceMLContext to establish our direct connection– it allows us to do inserts, selects, upserts, updates and many more functions without serialization

```python
%spark.pyspark
from splicemachine.spark.context import SpliceMLContext
splice = SpliceMLContext(spark)
```

**2. Load the Data into your database**
Next, we create the table in our Splice Machine database for our fraud data:
set schema cc_fraud;

drop table if exists cc_fraud_data;

cREATE TABLE cc_fraud.cc_fraud_data (
  time_offset INTEGER,
  v1 DOUBLE,
  v2 DOUBLE,
  v3 DOUBLE,
  v4 DOUBLE,
  v5 DOUBLE,
  v6 DOUBLE,
  v7 DOUBLE,
  v8 DOUBLE,
  v9 DOUBLE,
  v10 DOUBLE,
  v11 DOUBLE,
  v12 DOUBLE,
  v13 DOUBLE,
  v14 DOUBLE,
  v15 DOUBLE,
  v16 DOUBLE,
  v17 DOUBLE,
  v18 DOUBLE,
  v19 DOUBLE,
  v20 DOUBLE,
  v21 DOUBLE,
  v22 DOUBLE,
  v23 DOUBLE,
  v24 DOUBLE,
  v25 DOUBLE,
  v26 DOUBLE,
  v27 DOUBLE,
  v28 DOUBLE,
  amount DECIMAL(10,2),
  class_result INT
);

And then we import the data from S3 into the table:
call SYCS_UTIL.IMPORT_DATA (  
  'cc_fraud',  
  'cc_fraud_data',  
  null,  
  's3a://splice-demo/kaggle-fraud-data/creditcard.csv',  
  ',',  
  null,  
  null,  
  null,  
  null,  
  null,  
  100000,  
  's3a://splice-demo/kaggle-fraud-data/bad',  
  null,  
  null);

3. Try visualizing your data in the Notebook:
You can query the data and use one of the many visualizations built into Zeppelin to display your results. For example, you might run the following query to find the imbalance of fraud data, and display it as a pie chart:

```sql
%splicemachine
select class_result, count(*) from cc_fraud.cc_fraud_data group by class_result;
```
4. Create your MLManager Instance
To use ML Manager, you need to first create a class instance:

```python
%spark.pyspark
from splicemachine.ml.management import MLManager
manager = MLManager()
```

5. Create an Experiment
Now we'll create an MLflow experiment named `fraud_demo`:

```python
%spark.pyspark
manager.create_experiment('fraud-demo')
manager.set_active_experiment('fraud-demo')
```

6. Load our data into a DataFrame
And then we'll pull the data from our database table directly into a Spark DataFrame:

```python
df = splice.df("SELECT * FROM cc_fraud.cc_fraud_data")
df = df.withColumnRenamed('CLASS_RESULT', 'label')
z.show(df)
```

7. View your Experiment in the MLflow UI
You can now view your new experiment in the MLflow Tracking UI, at port 5001:

![MLflow UI](image)

Running Your First Experiment
Now that we're set up, let's create a run named Ben and run our experiment, using the logging functionality of MLManager to record and track the attributes of our run.

We'll use these steps to run our experiment:

1. Create a run
1. **Create a run**

We use a method of our `MLManager` object to create a new run:

```python
manager.create_new_run(user_id='Ben')
```

2. **Run the experiment**

We'll start our first MLflow run; since our data contains a limited number of fraudulent examples, we decide to expand that number for training purposes. To achieve this, we oversample fraudulent transactions and undersample non-fraudulent ones:

```python
#start our first MLflow run
manager.create_new_run(user_id='Ben')

#oversample fraud data 2X
fraud_data = df.filter('label=1')
print('fraud data has {} rows'.format(fraud_data.count()))
fraud_data = fraud_data.unionAll(fraud_data)
print('fraud data has {} rows'.format(fraud_data.count()))
#log oversample rate
manager.log_param('oversample','2X')

#undersample non-fraud data 1:1
non_fraud_df = df.filter('label=0')
ratio = float(fraud_data.count())/float(df.count())
sampled_non_fraud = non_fraud_df.sample(withReplacement=False, fraction=ratio)
final_df = fraud_data.unionAll(sampled_non_fraud)
#log undersample ratio
manager.log_param('undersample', '1:1')
z.show(final_df)
```

3. **Create a Pipeline**

Now we can create a Pipeline to normalize our continuous features. We'll use the `StandardScaler`, which standardizes features by scaling to unit variance and/or removing the mean using, column summary statistics on the samples in the training set.
And we’ll create our feature vector with the VectorAssembler transformer, which combines a given list of columns into a single vector column

```python
%spark.pyspark
from pyspark.ml.feature import StandardScaler, VectorAssembler
from pyspark.ml import Pipeline, PipelineModel

feature_cols = df.columns
feature_cols.remove("label")
feature_cols.remove('TIME_OFFSET')
print("Features: " + str(feature_cols))

assembler = VectorAssembler(inputCols=feature_cols, outputCol='features')
scaler = StandardScaler(inputCol="features", outputCol='scaledFeatures')
stages = [assembler, scaler]
#log preprocessing steps
manager.log_param('preprocessing','Pipeline[VectorAssembler, StandardScaler]')
#log features that we will use
manager.log_param('features',str(feature_cols))
pipeline = Pipeline(stages=stages)
z.show(pipeline.fit(df).transform(df))
```

### 4. Train and Run the Model

Now we can train and run this model using the SpliceBinaryClassificationEvaluator, again logging our parameters and metrics.
from pyspark.ml.classification import MultilayerPerceptronClassifier
from splicemachine.ml.utilities import SpliceBinaryClassificationEvaluator
import time

evaluator = SpliceBinaryClassificationEvaluator(spark)
first = len(feature_cols)
hidden = first/2
output = 2
layers = [first,hidden,output]
nn = MultilayerPerceptronClassifier(maxIter=100, layers=layers, blockSize=64, seed=5724, featuresCol='scaledFeatures')
manager.log_param('classifier', 'neural network')
manager.log_param('maxIter', '100')
manager.log_param('layers', '{{first}, {hidden}, 2}'.format(first=first,hidden=hidden))
manager.log_param('blockSize', '64')

df = df.repartition(50)
train, test = df.randomSplit([0.8,0.2])
t0 = time.time()
stages.append(nn)
full_pipeline = Pipeline(stages=stages)
model = full_pipeline.fit(train)
time_taken = time.time() - t0
print("Model took: " + str(time_taken) + " seconds to train")
#make predictions
predictions = model.transform(test)
evaluator.input(predictions)
z.show(evaluator.get_results())

#log how long the model took
manager.log_metric('time',time_taken)
#log metrics for reference
vals = evaluator.get_results('dict')
for key in vals:
    manager.log_metric(key, vals[key])

5. View Run Information
You can now view the run in the MLflow user interface, at port 5001:
6. Make Sure Model is Generalizable to Unbalanced Data

We have a model that looks fairly accurate; however, we trained this model on balanced data, so we need to verify that it can be generalized to work with unbalanced data:

```
%spark.pyspark
#pull in full dataset
new_df = splice.df('select * from cc_fraud.cc_fraud_data')
new_df = new_df.withColumnRenamed('CLASS_RESULT', 'label')

#transform and run model on new dataframe
new_predictions = model.transform(new_df)
new_eval = SpliceBinaryClassificationEvaluator(spark)
new_eval.input(new_predictions)
z.show(new_eval.get_results())
```

7. Save the Model

We want to be able to retrain our model when new data arrives, so we'll save the pipeline and model to an S3 bucket. And since we're planning to deploy this model, we'll also save it to MLflow:
%spark.pyspark
#save the pipeline and model to s3
model.save('s3a://splice-demo/fraudDemoPipelineModel')
#save model to MLflow for deployment
manager.log_spark_model(model)

**Trying a Different Model**

Now that we've saved our run, we can look at creating a different pipeline and comparing results; this time, we'll re-import the data and create a pipeline by oversampling at a 1.5x rate and using a LogisticRegression model, in these steps:

1. **Start a new run**
2. **Scale and vectorize our features**
3. **Train and test the model**
4. **Test on unbalanced Data**
5. **Compare results**
6. **Save the model**

**Start a new run**

First, we'll create a new run and name it Amy:

```python
manager.create_new_run(user_id=`Amy``)
```

Next we'll reload the data from our database table into a Spark DataFrame, and then and undersample/oversample like we did previously:
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import StandardScaler, VectorAssembler
from pyspark.ml import Pipeline, PipelineModel

# create new run
manager.create_new_run(user_id='Amy')
df = splice.df("SELECT * FROM cc_fraud.cc_fraud_data")
df = df.withColumnRenamed('CLASS_RESULT', 'label')

# oversample fraud data 1.5X
fraud_data = df.filter('label=1')
print('fraud data has {} rows'.format(fraud_data.count()))

# sample half the data
fraud_ratio = 0.5
half_fraud_data = fraud_data.sample(withReplacement=False, fraction=fraud_ratio)

# 1.5X as many rows
fraud_data = fraud_data.unionAll(half_fraud_data)
print('fraud data has {} rows'.format(fraud_data.count()))

# log oversample rate
manager.log_param('oversample', '1.5X')

# undersample non-fraud data 1:1
non_fraud_df = df.filter('label=0')

ratio = float(fraud_data.count()) / float(df.count())
sampled_non_fraud = non_fraud_df.sample(withReplacement=False, fraction=ratio)

final_df = fraud_data.unionAll(sampled_non_fraud)

# log undersample ratio
manager.log_param('undersample', '1:1')

2. Scale and Vectorize our Features
We'll again use the StandardScaler and VectorAssembler components to normalize and vectorize our features:
%spark.pyspark
#feature engineering
feature_cols = df.columns
feature_cols.remove("label")
print("Features: " + str(feature_cols))

#feature vector and scale features
assembler = VectorAssembler(inputCols=feature_cols, outputCol='features')
scaler = StandardScaler(inputCol="features", outputCol='scaledFeatures')
stages = [assembler, scaler]
#log preprocessing steps
manager.log_param('preprocessing','Pipeline[VectorAssembler, StandardScaler]')
#log features that we will use
for feature,i in zip(feature_cols,range(len(feature_cols))):
    manager.log_param('feature {}'.format(i),feature)

3. Train and Test the Model
Now we can train and test this model:

%spark.pyspark
#build and evaluate model
evaluator = SpliceBinaryClassificationEvaluator(spark)
lr = LogisticRegression(featuresCol='scaledFeatures')
manager.log_param('classifier', 'logistic regression')
stages.append(lr)

df = df.repartition(50)
train, test = df.randomSplit([0.8,0.2])
t0 = time.time()
full_pipeline = Pipeline(stages=stages)
model = full_pipeline.fit(train)
time_taken = time.time() - t0
print("Model took: " + str(time_taken) + " seconds to train")
#make predictions
predictions = model.transform(test)
evaluator.input(predictions)
z.show(evaluator.get_results())

#log how long the model took
manager.log_metric('time',time_taken)
#log metrics for reference
vals = evaluator.get_results('dict')
for key in vals:
    manager.log_metric(key, vals[key])

4. Test on Unbalanced Data
We also need to make sure that this model will generalize to work with unbalanced data:
5. Compare Results

We can now visit the MLflow Tracking UI again to compare the results of this run with our previous one:

<table>
<thead>
<tr>
<th>Date</th>
<th>Name</th>
<th>Time</th>
<th>Model Details</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-02-21 Ben</td>
<td>16:16:27</td>
<td>TIME_OFFSET, 'V1', 'V2', 'V3', 'V4', classifier: neural network layers: [30, 15, 2] maxiter: 100 oversample: 2x preprocessing: Pipeline(VectorAssembler, StandardScaler) undersample: 1:1</td>
<td>ACC: 1 F1: 0.883 FDR: 0.16 FNR: 0.071 FPR: 0 MCC: 0.884 NPV: 1 PPV: 0.94 SPC: 0.929 TPR: 0.929 time: 107.5</td>
<td></td>
</tr>
</tbody>
</table>

Though this run was faster, its was not as accurate; the False Positive Rate (FPR) was too high to use for fraud prediction, so we'll move forward with our initial model.
6. Save the run
We'll save this run to S3 for future testing; however, since we won’t be deploying it, we don't need to log it to MLflow at this time.

```python
%spark.pyspark
model.save('s3a://splice-demo/fraudDemoPipelineLogisticRegression')
```

**Deploying the Model with SageMaker**

Once you've run an experiment run that looks good, you can interface with Amazon SageMaker to deploy your model on AWS, following these steps:

1. [Create an ECR Repository for your Experiment](#).
2. [Find your Experiment and Run IDs](#).
3. [Deploy Your Model](#).

**Step 1: Create an ECR Repository**

Elastic Container Registry (ECR) is Amazon's managed AWS Docker registry service; it supports private Docker repositories with resource-based permissions using AWS IAM so that specific users or Amazon EC2 instances can access repositories and images.

When you tell the Splice ML Manager to deploy your model to SageMaker, ML Manager creates a Docker image, saves it to your ECR repo, and tells AWS to deploy it for you. You then have an endpoint on AWS with a RESTfull API that your apps can use to interact with your model.

To take advantage of this capability, you need to create a repository on ECR. See the [Amazon ECR documentation](#) for information about creating your repository.

**Step 2: Find your Experiment and Run IDs**

Before deploying your model, you need to have the IDs of the experiment and run that you want to deploy; you can find both of these in the MLflow Tracking UI. Follow these steps:

1. Navigate to port 5001 in your web browser to display the MLflow Tracking UI. For example: https://myacct-machine.splicemachine.io:5001/#!/.
2. Select the experiment that you want to deploy. In this example, we've selected the experiment named `test_exp`:

3. Record the Experiment ID displayed for the experiment; in the above example, we're viewing Experiment ID 1.
4. Select the ID of the run that you want to deploy; here we've selected the topmost (most recent) run of Experiment 1. When you click this Run ID, you'll see its details displayed:

5. Copy the Run ID value to your clipboard.

**Step 3: Deploy Your Model**

Once you know your Experiment and Run ID values, you can use the Splice ML Jobs Tracker to deploy your model. Follow these steps:

1. Navigate to port 5003 in your web browser to display the *ML Manager* Jobs Tracker. For example: https://myacct-machine.splicemachine.io:5003/#/

2. Click the **deploy** link at the top of the screen to display the deploy form:
3. Fill in the form fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run UUID (can be found in MLFlow UI)</td>
<td>The Run ID that you copied to your clipboard from the MLflow Tracking UI.</td>
</tr>
<tr>
<td>Run Experiment ID (can be found in MLFlow UI)</td>
<td>The ID of the experiment that you recorded from the MLflow Tracking UI.</td>
</tr>
<tr>
<td>SageMaker App Name When Deployed</td>
<td>The name you want to use for your deployed App.</td>
</tr>
<tr>
<td>AWS Region</td>
<td>The AWS regions in which you want the app deployed. Select one of the values from the drop-down list.</td>
</tr>
<tr>
<td>Deployment Mode</td>
<td></td>
</tr>
<tr>
<td>Instance Count</td>
<td></td>
</tr>
<tr>
<td>SageMaker Instance Type</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Deployment Mode.</td>
<td>Select one of the values from the drop-down list:</td>
</tr>
<tr>
<td></td>
<td>Create Create a new deployment.</td>
</tr>
<tr>
<td></td>
<td>Replace Replace an existing deployment.</td>
</tr>
<tr>
<td></td>
<td>Add ??????????????????????????????????</td>
</tr>
<tr>
<td>Instance Count</td>
<td>The number of instances that you want deployed.</td>
</tr>
<tr>
<td>SageMaker Instance Type</td>
<td>The AWS instance type that you want to use for your deployment.</td>
</tr>
<tr>
<td></td>
<td>Select one of the values from the drop-down list.</td>
</tr>
</tbody>
</table>
Here’s an example of a completed deployment form:

**Deploy to SageMaker**

Please create an ECR repository called mlflow-pyfunc in the SageMaker AWS Region you are deploying before submitting this form

- d7ca936fbcec425ca1cc631ffa6cbec7
- 1
- MyMLApp
- us-east-2
- Create
- 1
- ml.m5.xlarge

**SUBMIT**

4. Click the **Submit** button to deploy your model.

**Retraining the Model with New Data**

Whenever additional labeled data arrives, we can pull either or both of our models from S3 and run the new data through it, allowing us to easily enhance accuracy over time.
Redeploying the Model

To redeploy your model after retraining, you can use the same steps you used when originally deploying it, as described in [Deploy Your Model](#), above. Simply select Replace as your deployment mode, and your model will be redeployed.
The Splice ML Manager API

This topic describes the methods available in the ML Manager API; you'll find examples of each of these in the Using ML Manager topic, in this chapter.

You can use Splice ML Manager with Python in Zeppelin notebooks, using our pyspark interpreter and our MLManager class in your program to manipulate experiments.

This topic contains the following sections:

- Getting Started with the ML Manager API
- ML Manager Methods

Getting Started with the ML Manager API

To get started with MLManager, you need to:

1. Create your MLManager instance
2. Establish a connection to your database
3. Create an experiment
4. Create a run
5. Run your experiment(s)

1. Create your MLManager instance

To use ML Manager, you need to first create a class instance:

```python
%spark.pyspark
from splicemachine.ml.management import MLManager
manager = MLManager()
```

2. Connect to Your Database

You can establish a connection to your database using our Native Spark Datasource, which is encapsulated in the SpliceMLContext object. Once you've connected, you can use the SpliceMLContext to perform database inserts, selects, upserts, updates and many more functions, all directly from Spark, without any required serialization.

```python
%spark.pyspark
from splicemachine.spark.context import SpliceMLContext
splice = SpliceMLContext(spark)
```

3. Create an Experiment

This code creates a new experiment and sets it as the active experiment:
4. Create a Run
We use a method of our MLManager object to create a new run:

```python
manager.create_new_run(user_id='firstrun')
```

Run Your Experiment
The Using ML Manager topic in this chapter provides a complete example of running machine learning experiments with ML Manager in a Zeppelin notebook.

ML Manager Methods
This section describes the methods of the MLManager class, in these three subsections:

- **Experiment and Run Methods**
- **Logging Methods**
- **Tagging Methods**

Experiment and Run Methods
This section describes the ML Manager methods for working with experiments and runs:

- `create_experiment`
- `create_new_run`
- `reset_run`
- `set_active_experiment`
- `set_active_run`

**create_experiment**
Use the `create_experiment` method to create and name an experiment.

```python
manager.create_experiment( experiment_name )
```

`experiment_name`
A string name or integer ID you want to use for the experiment.

Example:
manager.create_experiment( 'myFirstExperiment')

**create_new_run**

Use the `create_new_run` to create a new run under the currently active experiment, and to make the new run the currently active run.

```
manager.create_new_run( run_name )
```

*run_name*

The name of the user creating the run.

Example:

```
manager.create_new_run( 'myNewRun')
```

**reset_run**

Use the `reset_run` method to rest the current run. This deletes logged parameters, metrics, artifacts, and other information associated with the run.

```
manager.reset_run( )
```

Example:

```
manager.reset_run( )
```

**set_active_experiment**

Use the `set_active_experiment` method to make an existing experiment the active experiment. All new runs will be created under this experiment.

```
manager.set_active_experiment( experiment_name )
```

*experiment_name*

A string name you want to use for the experiment.

Example:

```
manager.set_active_experiment( 'myFirstExperiment')
```
**set_active_run**

Use the `set_active_run` method to set a previous run as the active run under the current experiment; this allows you to log metadata for a completed run.

```python
manager.set_active_run( run_name )
```

**run_name**

The string name of the run you want to make the active run.

Example:

```python
manager.set_active_run( 'myNewRun' )
```

---

**Logging Methods**

This section describes the ML Manager methods for logging models, parameters, metrics, and artifacts:

- `log_artifact`
- `log_artifacts`
- `log_metric`
- `log_model`
- `log_param`
- `log_spark_model`

**log_artifact**

Use the `log_artifact` method to log a local file or directory as an artifact of the currently active run.

```python
manager.log_artifact( local_path, artifact_path )
```

**local_path**

The path to the file that you want written to your artifacts URI.

**artifact_path**

Optional. The subdirectory of your artifacts URI to which you want the artifact written.

Example:

```python
manager.log_artifact( '/tmp/myRunData' )
```
log_artifacts
Use the log_artifacts method to log the contents of a local directory as artifacts of the currently active run.

```python
manager.log_artifacts( local_dir, artifact_path )
```

**local_dir**
The path to the directory of files that you want written to your artifacts URI.

**artifact_path**
Optional. The subdirectory of your artifacts URI to which you want the artifact written.

Example:

```python
manager.log_artifacts( '/tmp/myRunInfo' )
```

log_metric
Use the log_metric method to log a (key, numeric-value) pair for the currently active run. You can update a metric throughout the course of the run, and you can subsequently view the metric's history.

```python
manager.log_metric( metric_name, metric_value )
```

**metric_name**
A string naming the metric to log.

**metric_value**
The double-precision numeric value to log for the metric.

Example:

```python
#log how long the model took
manager.log_metric('time', time_taken)
```

log_model
Use the log_model method to log a model for the currently active run.

```python
log_model( model, module )
```

**model**
The fitted pipeline/model (in Spark) that you want to log.

**module**
The module that the model is part of; for example, mlflow.spark or mlflow.sklearn.
Example:

```python
#save model to MLflow for deployment
manager.log_model(model, 'mlflow.sklearn')
```

**log_param**
Use the `log_param` method to log a (key, string-value) pair for the currently active run.

```python
manager.log_param(param_name, param_value)
```

*param_name*
A string naming the parameter to log.

*param_value*
The string value to log for the parameter.

Example:

```python
manager.log_param('classifier', 'neural network')
manager.log_param('maxIter', '100')
```

**log_spark_model**
Use the `log_spark_model` method to save a MLlib model you've created to MLflow, for future deployment.

```python
log_spark_model(model)
```

*model*
The fitted pipeline/model you want to log.

Example:

```python
#save the pipeline and model to s3
model.save('s3a://myModels/myFirstModel')
#save model to MLflow for deployment
manager.log_spark_model(model)
```

**Tagging Methods**
This section describes the ML Manager methods for tagging:

- `set_tag`
**set_tag**

Use the `set_tag` method to set the value of a tag for the current run. Tags are specific pieces of information associated with a run, such as the project ID, the version ID, or the deployable status.

```python
set_tag( key, value )
```

- **tag_name**
  The name of the tag you want to assign a value to for the current run.

- **tag_value**
  The string value to for the tag.

Example:

```python
manager.set_tag('projectId', 'myNewProject')
```
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:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifiers</td>
<td>Describes the different identifiers used in SQL.</td>
</tr>
<tr>
<td>Data Types</td>
<td>Describes the data types used in SQL.</td>
</tr>
<tr>
<td>Statements</td>
<td>Reference pages for our implementation of each SQL statement.</td>
</tr>
<tr>
<td>Clauses</td>
<td>Reference pages for our implementation of SQL clauses.</td>
</tr>
<tr>
<td>Expressions</td>
<td>Describes the expressions you can use in SQL.</td>
</tr>
<tr>
<td>Join Operations</td>
<td>Reference pages for our implementation of SQL join operations.</td>
</tr>
<tr>
<td>Queries</td>
<td>Reference pages for our implementation of SQL queries.</td>
</tr>
<tr>
<td>SQL Built-in Functions</td>
<td>Reference pages for the standard SQL functions featured in our implementation.</td>
</tr>
<tr>
<td>System Tables</td>
<td>Descriptions of the system tables.</td>
</tr>
<tr>
<td>Argument Matching</td>
<td>Describes how Splice Machine matches Java data types and methods with arguments supplied in SQL statements.</td>
</tr>
<tr>
<td>SQL Limitations</td>
<td>A summary of various size limitations in Splice Machine.</td>
</tr>
<tr>
<td>Reserved Words</td>
<td>A list of the reserved words in Splice Machine.</td>
</tr>
</tbody>
</table>

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.
**ABS or ABSVAL**

ABS or ABSVAL returns the absolute value of a numeric expression.

**Syntax**

```sql
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**

```sql
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

**See Also**

- CURRENT_DATE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- DATE type
- DATE function
- DAY
- EXTRACT
- LASTDAY
- MONTH
- MONTH_BETWEEN
- MONTHNAME
- NEXTDAY
NOW

QUARTER

TIME type

TIME function

TIMESTAMP type

TIMESTAMP function

TO_CHAR

TO_DATE

WEEK

Working with Dates
**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**

```
slice> 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**

\[
\text{ATAN2} ( \ y, \ x )
\]

\( y \)

A DOUBLE PRECISION number.

\( x \)

A DOUBLE PRECISION number.

**Results**

ATAN2 returns the arctangent 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:

```sql
--- 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:
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;

<table>
<thead>
<tr>
<th>POS&amp;</th>
<th>ID</th>
<th>SALARY</th>
<th>Average for Po&amp;</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1</td>
<td>17277777</td>
<td>3733139.8000</td>
</tr>
<tr>
<td>C</td>
<td>13</td>
<td>468674</td>
<td>3733139.8000</td>
</tr>
<tr>
<td>C</td>
<td>18</td>
<td>800000</td>
<td>3733139.8000</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
<td>41598</td>
<td>3733139.8000</td>
</tr>
<tr>
<td>C</td>
<td>24</td>
<td>77650</td>
<td>3733139.8000</td>
</tr>
<tr>
<td>IF</td>
<td>23</td>
<td>91516</td>
<td>91516.0000</td>
</tr>
<tr>
<td>1B</td>
<td>2</td>
<td>3600000</td>
<td>1815252.5000</td>
</tr>
<tr>
<td>1B</td>
<td>26</td>
<td>30505</td>
<td>1815252.5000</td>
</tr>
<tr>
<td>LF</td>
<td>6</td>
<td>4000000</td>
<td>1792987.0000</td>
</tr>
<tr>
<td>LF</td>
<td>16</td>
<td>278961</td>
<td>1792987.0000</td>
</tr>
<tr>
<td>LF</td>
<td>27</td>
<td>1100000</td>
<td>1792987.0000</td>
</tr>
<tr>
<td>P</td>
<td>28</td>
<td>6950000</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>29</td>
<td>485314</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>30</td>
<td>4000000</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>31</td>
<td>12000000</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>32</td>
<td>9000000</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>33</td>
<td>18000000</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>34</td>
<td>20833333</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>35</td>
<td>3578825</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>36</td>
<td>2100000</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>37</td>
<td>210765</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>38</td>
<td>507500</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>39</td>
<td>507500</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>40</td>
<td>6000000</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>41</td>
<td>6000000</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>42</td>
<td>374385</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>43</td>
<td>4000000</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>44</td>
<td>72103</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>45</td>
<td>91516</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>46</td>
<td>5000000</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>47</td>
<td>74877</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>P</td>
<td>48</td>
<td>163620</td>
<td>4759511.3333</td>
</tr>
<tr>
<td>3B</td>
<td>5</td>
<td>509000</td>
<td>2654500.0000</td>
</tr>
<tr>
<td>3B</td>
<td>14</td>
<td>4800000</td>
<td>2654500.0000</td>
</tr>
<tr>
<td>MI</td>
<td>15</td>
<td>288767</td>
<td>288767.0000</td>
</tr>
<tr>
<td>SS</td>
<td>4</td>
<td>3175000</td>
<td>3175000.0000</td>
</tr>
<tr>
<td>RF</td>
<td>8</td>
<td>18500000</td>
<td>9166666.6666</td>
</tr>
<tr>
<td>RF</td>
<td>10</td>
<td>1000000</td>
<td>9166666.6666</td>
</tr>
<tr>
<td>RF</td>
<td>12</td>
<td>8000000</td>
<td>9166666.6666</td>
</tr>
<tr>
<td>2B</td>
<td>3</td>
<td>507500</td>
<td>339719.5000</td>
</tr>
<tr>
<td>2B</td>
<td>11</td>
<td>171939</td>
<td>339719.5000</td>
</tr>
<tr>
<td>CF</td>
<td>7</td>
<td>10250000</td>
<td>10250000.0000</td>
</tr>
<tr>
<td>UT</td>
<td>17</td>
<td>1450000</td>
<td>749959.0000</td>
</tr>
<tr>
<td>UT</td>
<td>22</td>
<td>49918</td>
<td>749959.0000</td>
</tr>
</tbody>
</table>
### 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;

<table>
<thead>
<tr>
<th>ID</th>
<th>TotalBases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>262</td>
</tr>
<tr>
<td>2</td>
<td>235</td>
</tr>
<tr>
<td>3</td>
<td>174</td>
</tr>
<tr>
<td>4</td>
<td>234</td>
</tr>
<tr>
<td>5</td>
<td>245</td>
</tr>
<tr>
<td>6</td>
<td>135</td>
</tr>
<tr>
<td>7</td>
<td>170</td>
</tr>
<tr>
<td>8</td>
<td>99</td>
</tr>
<tr>
<td>9</td>
<td>135</td>
</tr>
<tr>
<td>10</td>
<td>85</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>TYPES</th>
<th>BOOLEAN</th>
<th>TINYINT</th>
<th>SMALLINT</th>
<th>INTEGER</th>
<th>DECIMAL</th>
<th>REAL</th>
<th>DOUBLE</th>
<th>FLOAT</th>
<th>CHAR</th>
<th>VARCHAR</th>
<th>VARCHAR</th>
<th>CLOB</th>
<th>DLOB</th>
<th>DATE</th>
<th>TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TINYINT</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>INTEGER</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
### Type Categories

This section lists information about converting specific data types. The Splice Machine ANSI SQL data types are categorized as follows:

If a conversion is valid, CASTs are allowed. Size incompatibilities between the source and target types might cause runtime errors.

<table>
<thead>
<tr>
<th>TYPES</th>
<th>BOOLEAN</th>
<th>TINYINT</th>
<th>SMALLINT</th>
<th>INTEGER</th>
<th>BIGINT</th>
<th>DECIMAL</th>
<th>REAL</th>
<th>DOUBLE</th>
<th>FLOAT</th>
<th>CHAR</th>
<th>VARCHAR</th>
<th>LONG VARCHAR</th>
<th>CLOB</th>
<th>BLOB</th>
<th>DATE</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>Y Y Y Y Y Y Y Y - - - - -</td>
<td>DECIMAL</td>
<td>Y Y Y Y Y Y Y Y - - - - -</td>
<td>REAL</td>
<td>Y Y Y Y Y Y Y - - - - -</td>
<td>DOUBLE</td>
<td>Y Y Y Y Y Y Y - - - - -</td>
<td>FLOAT</td>
<td>Y Y Y Y Y Y Y - - - - -</td>
<td>CHAR</td>
<td>Y Y Y Y Y - - - Y Y Y -</td>
<td>VARCHAR</td>
<td>Y Y Y Y Y - - - Y Y Y -</td>
<td>LONG VARCHAR</td>
<td>Y - - - - - Y Y Y - - - -</td>
<td>CLOB</td>
</tr>
</tbody>
</table>
### 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:

```sql
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 \(\text{INTEGER}\) 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 \(\text{TIMESTAMP}\).

If a \(\text{DATE}\) is converted to a \(\text{TIMESTAMP}\), the \(\text{TIME}\) component of the resulting \(\text{TIMESTAMP}\) is always \(00:00:00\).

If a \(\text{TIME}\) data value is converted to a \(\text{TIMESTAMP}\), the \(\text{DATE}\) component is set to the value of \(\text{CURRENT\_DATE}\) at the time the CAST is executed.

If a \(\text{TIMESTAMP}\) is converted to a \(\text{DATE}\), the \(\text{TIME}\) component is silently truncated.

If a \(\text{TIMESTAMP}\) is converted to a \(\text{TIME}\), the \(\text{DATE}\) component is silently truncated.

**Implicit Conversions and Joins**

Splice Machine performs implicit type conversion when performing joins to allow joining of mismatched column types. Specifically, when joining a \(\text{CHAR}\) or \(\text{VARCHAR}\) column on a column of the following types, an attempt is made to convert the string value into that column’s type:

- \(\text{BOOLEAN}\)
- \(\text{DATE}\)
- \(\text{TIME}\)
- \(\text{TIMESTAMP}\)

If any row in a query that involves implicit type conversion contains a column value cannot be converted to the desired type, the join fails, and the following error is thrown:

```
ERROR 22007: The syntax of the string representation of a datetime value is incorrect
```

**Examples**

Here are a few explicit type conversions:
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);

Here's an example of implicit conversion failures during attempted joins:

CREATE TABLE s1 (a1 INT, b1 VARCHAR(16));
CREATE TABLE s2 (a2 INT, b2 DATE);
CREATE TABLE s3 (a3 INT, b3 BOOLEAN);

INSERT INTO s2 VALUES (2,'2018-11-11');
INSERT INTO s3 VALUES (3,true);
INSERT INTO s1 VALUES(1,'2018-11-11');

SELECT a1 FROM s1 JOIN s2 ON  b1 = b2;
A1
  -------
   1

INSERT INTO s1 VALUES(2,'foo');
SELECT a1 FROM s1 JOIN s2 ON  b1 = b2;
ERROR 22007: The syntax of the string representation of a datetime value is incorrec t.

DELETE FROM s1;
INSERT INTO s1 VALUES (1,'1');
SELECT a1 FROM s1 JOIN s3 ON  b1 = b3;
A1
  -------
   1

INSERT INTO s1 VALUES(2,'foo');
SELECT a1 FROM s1 JOIN s3 ON  b1 = b3;
ERROR 22018: Invalid character string format for type BOOLEAN.
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 TINYINT, 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**

```sql
CHAR (DatetimeExpression )
```

*DatetimeExpression*

An expression that is one of the following three data types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>The result is the character representation of the date. The length of the result is 10.</td>
</tr>
<tr>
<td>TIME</td>
<td>The result is the character representation of the time. The length of the result is 8.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>The result is the character string representation of the timestamp. The length of the result is 26.</td>
</tr>
</tbody>
</table>

**Decimal to character**

```sql
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**

```sql
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
The **CHR** function returns the ASCII character equivalent of a numeric ASCII code value, as a single byte value.

**Syntax**

```
CHR( number)
```

*number*  
The ASCII code value for a character.

**Results**

This function takes as an argument a numeric value, or any value that can be implicitly converted to numeric, and returns a character. The character returned is described in the following table:

<table>
<thead>
<tr>
<th>Value of number</th>
<th>Returned value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 255</td>
<td>The ASCII character corresponding to the value of <code>number</code>.</td>
</tr>
<tr>
<td>&gt; 255</td>
<td>The ASCII character corresponding to the value of <code>number mod 256</code>.</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>The ASCII character corresponding to the value 255.</td>
</tr>
<tr>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Examples

splice> VALUES( Chr(65) );
1
----
A

splice> VALUES( Chr(321) );
1
----
A

splice> VALUES( Chr(null) );
1
----
NULL
**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:

```sql
COALESCE( expression_1, expression_2,...expression_n);
```

is equivalent to:

```sql
CASE WHEN expression_1 IS NOT NULL THEN expression_1
    ELSE WHEN expression_1 IS NOT NULL THEN expression_2
   ...
    ELSE expression_n;
```

**Syntax**

```sql
COALESCE ( expression1, expression2 [, expressionN]* )
```

- **expression1**
  - An expression.

- **expression2**
  - 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

```sql
-- 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;

<table>
<thead>
<tr>
<th>ID</th>
<th>FldGames</th>
<th>PassedBalls</th>
<th>WildPitches</th>
<th>Pickoffs</th>
<th>FirstNonNull</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>142</td>
<td>4</td>
<td>20</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>131</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>99</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>4</td>
<td>140</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>5</td>
<td>142</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>6</td>
<td>88</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>7</td>
<td>124</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>8</td>
<td>51</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>9</td>
<td>93</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>10</td>
<td>79</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>39</td>
<td>73</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>52</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>41</td>
<td>70</td>
<td>NULL</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>42</td>
<td>55</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>43</td>
<td>77</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>46</td>
<td>67</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>134</td>
<td>4</td>
<td>34</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>119</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>51</td>
<td>147</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>52</td>
<td>148</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>53</td>
<td>152</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>54</td>
<td>64</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>55</td>
<td>93</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>56</td>
<td>147</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>57</td>
<td>85</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>58</td>
<td>62</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>59</td>
<td>64</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>62</td>
<td>53</td>
<td>1</td>
<td>11</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>64</td>
<td>59</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>81</td>
<td>76</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>82</td>
<td>71</td>
<td>NULL</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>84</td>
<td>68</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>81</td>
<td>NULL</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
```

33 rows selected
**Concatenation Operator**

The concatenation operator, which you can specify using either the symbol `||` or the word `CONCAT`, 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 } |
{ CharacterExpression  CONCAT  CharacterExpression } |
{ BitExpression  ||  BitExpression } |
{ BitExpression  CONCAT  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

```sql
-- returns 'San Francisco Giants'
splice> VALUES 'San' || ' ' || 'Francisco' || ' ' || 'Giants';

-- returns NULL
splice> VALUES CAST (null AS VARCHAR(7)) CONCAT '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
The COS function returns the cosine of a specified number.

Syntax

```sql
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

```sql
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
- RADIANS 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
- **RADIANS** function
- **SIN** function
- **SINH** function
- **TAN** function
- **TANH** function
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
- RADIANS 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;

<table>
<thead>
<tr>
<th>Players</th>
<th>TEAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>Cards</td>
</tr>
<tr>
<td>48</td>
<td>Giants</td>
</tr>
</tbody>
</table>

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 ROW)
 as "Running Count"
FROM Players JOIN Batting ON Players.ID=Batting.ID
WHERE homeRuns > 5
ORDER BY "Running Count";

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>HOMER&amp;</th>
<th>Running Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jeremy Packman</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Jason Minman</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Stan Post</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>John Purser</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Harry Pennello</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>Kelly Wacherman</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>Mitch Duffer</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>Michael Rastono</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>Jack Hellman</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>Jonathan Pearlman</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>Roger Green</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>Billy Bopper</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>Buddy Painter</td>
<td>19</td>
<td>13</td>
</tr>
<tr>
<td>Bob Cranker</td>
<td>21</td>
<td>14</td>
</tr>
<tr>
<td>Mitch Canepa</td>
<td>28</td>
<td>15</td>
</tr>
</tbody>
</table>

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**

```sql
splice> VALUES(CURRENT SCHEMA);
1
----------------------------------------
SPLICE
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 that 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';

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>BIRTHDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yuri Milleton</td>
<td>1982-07-13</td>
</tr>
<tr>
<td>Jonathan Pearlman</td>
<td>1982-05-28</td>
</tr>
<tr>
<td>David Janssen</td>
<td>1979-08-10</td>
</tr>
<tr>
<td>Jason Larrimore</td>
<td>1978-10-23</td>
</tr>
<tr>
<td>Tam Croonster</td>
<td>1980-12-19</td>
</tr>
<tr>
<td>Alex Wister</td>
<td>1981-08-30</td>
</tr>
<tr>
<td>Robert Cohen</td>
<td>1975-09-05</td>
</tr>
<tr>
<td>Mitch Brandon</td>
<td>1980-06-06</td>
</tr>
</tbody>
</table>

8 rows selected
See Also

- CURRENT_TIME
- CURRENT_TIMESTAMP
- DATE type
- DATE function
- DAY
- EXTRACT
- LASTDAY
- MONTH
- MONTH_BETWEEN
- MONTHNAME
- NEXTDAY
- NOW
- QUARTER
- TIME type
- TIME function
- TIMESTAMP type
- TIMESTAMP function
- TO_CHAR
- TO_DATE
- WEEK
- Working with Dates
CURRENT_ROLE returns a list of role names for the current user. If there is no current role, it returns NULL.

This function returns a string of up to 32672 characters.

Syntax

```
CURRENT_ROLE
```

Example

```sql
call syscs_util.syscs_create_user('jdoe', 'jdoe');
create schema test1;
create role admin1;
grant all privileges on schema test1 to admin1;
create schema test2;
create role admin2;
grant all privileges on schema test2 to admin2;
create schema test3;
create role admin3;
grant all privileges on schema test3 to admin3;

grant admin1 to jdoe;
grant admin2 to jdoe;
grant admin3 to jdoe;

connect 'jdbc:splice://localhost:1527/splicedb;user=jdoe;password=jdoe' as jdoe_con;
splice> values current_role;
1
"ADMIN1", "ADMIN2", "ADMIN3"

1 row selected
```

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.

Splice Machine displays `TIME` and `TIMESTAMP` values using the current time zone for the server to which you are connected. `DATE` values, however, are stored as fixed strings, and are not affected by the time zone of the server to which you are connected.

**Syntax**

- `CURRENT_TIME`
- `CURRENT TIME`

or, alternately

**Results**
A time value.

**Examples**

```sql
splice> VALUES CURRENT_TIME;
1
--------
11:02:57
1 row selected
```

**See Also**

- `CURRENT_DATE`
- `CURRENT_TIMESTAMP`
Working with Dates
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.

Splice Machine displays `TIME` and `TIMESTAMP` values using the current time zone for the server to which you are connected. `DATE` values, however, are stored as fixed strings, and are not affected by the time zone of the server to which you are connected.

**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

**See Also**

› CURRENT_DATE

› CURRENT_TIME
DATE type
DATE function
DAY
EXTRACT
LASTDAY
MONTH
MONTH_BETWEEN
MONTHNAME
NEXTDAY
NOW
QUARTER
TIME type
TIME function
TIMESTAMP type
TIMESTAMP function
TO_CHAR
TO_DATE
WEEK
Working with Dates
**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
---------------------------------------------------------------
SPLICE
```

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 `yyyynnn`, where `yyyy` is a four-digit year value, and `nnn` is a three-digit day value in the range 001 to 366.

**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 ID < 20;
DaysSinceJFK
-----------
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
› CURRENT_TIME
› DATE type
› DAY
› EXTRACT
› LASTDAY
› MONTH
› MONTH_BETWEEN
› MONTHNAME
› NEXTDAY
› NOW
› QUARTER
TIME type
TIME function
TIMESTAMP type
TIMESTAMP function
TO_CHAR
TO_DATE
WEEK
Working with Dates
**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-Birth
-----------
  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
- CURRENT_TIME
- DATE type
- DATE function
- EXTRACT
- LASTDAY
- MONTH
- MONTH_BETWEEN
- MONTHNAME
- NEXTDAY
- NOW
» QUARTER
» TIME type
» TIME function
» TIMESTAMP type
» TIMESTAMP function
» TO_CHAR
» TO_DATE
» WEEK

» Working with Dates
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**

```sql
DEGREES ( number )
```

*number*

A **DOUBLE PRECISION** number that specifies the angle you want converted, in radians.

**Example**

```sql
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
- RADIANS 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**

```sql
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:
<table>
<thead>
<tr>
<th>Value</th>
<th>RANK</th>
<th>DENSE_RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>e</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

**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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>TEAM</th>
<th>SEASON</th>
<th>SALARY</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mitch Hassleman</td>
<td>Cards</td>
<td>2015</td>
<td>17000000</td>
<td>1</td>
</tr>
<tr>
<td>Yuri Milleton</td>
<td>Cards</td>
<td>2015</td>
<td>15200000</td>
<td>2</td>
</tr>
<tr>
<td>James Grasser</td>
<td>Cards</td>
<td>2015</td>
<td>9375000</td>
<td>3</td>
</tr>
<tr>
<td>Jack Hellman</td>
<td>Cards</td>
<td>2015</td>
<td>8300000</td>
<td>4</td>
</tr>
<tr>
<td>Larry Lintos</td>
<td>Cards</td>
<td>2015</td>
<td>7000000</td>
<td>5</td>
</tr>
<tr>
<td>Jeremy Johnson</td>
<td>Cards</td>
<td>2015</td>
<td>4125000</td>
<td>6</td>
</tr>
<tr>
<td>Mitch Canepa</td>
<td>Cards</td>
<td>2015</td>
<td>3750000</td>
<td>7</td>
</tr>
<tr>
<td>Mitch Brandon</td>
<td>Cards</td>
<td>2015</td>
<td>3500000</td>
<td>8</td>
</tr>
<tr>
<td>Robert Cohen</td>
<td>Cards</td>
<td>2015</td>
<td>3000000</td>
<td>9</td>
</tr>
<tr>
<td>James Woegren</td>
<td>Cards</td>
<td>2015</td>
<td>2675000</td>
<td>10</td>
</tr>
<tr>
<td>Sam Culligan</td>
<td>Cards</td>
<td>2015</td>
<td>2652732</td>
<td>11</td>
</tr>
<tr>
<td>Barry Morse</td>
<td>Cards</td>
<td>2015</td>
<td>2379781</td>
<td>12</td>
</tr>
<tr>
<td>Michael Rastono</td>
<td>Cards</td>
<td>2015</td>
<td>2000000</td>
<td>13</td>
</tr>
<tr>
<td>Carl Vanamos</td>
<td>Cards</td>
<td>2015</td>
<td>2000000</td>
<td>13</td>
</tr>
<tr>
<td>Alex Wister</td>
<td>Cards</td>
<td>2015</td>
<td>1950000</td>
<td>14</td>
</tr>
<tr>
<td>Pablo Bonjourno</td>
<td>Cards</td>
<td>2015</td>
<td>1650000</td>
<td>15</td>
</tr>
<tr>
<td>Jonathan Pearlman</td>
<td>Cards</td>
<td>2015</td>
<td>1500000</td>
<td>16</td>
</tr>
<tr>
<td>Jan Bromley</td>
<td>Cards</td>
<td>2015</td>
<td>1200000</td>
<td>17</td>
</tr>
<tr>
<td>Martin Cassman</td>
<td>Giants</td>
<td>2015</td>
<td>20833333</td>
<td>1</td>
</tr>
<tr>
<td>Harry Pennello</td>
<td>Giants</td>
<td>2015</td>
<td>18500000</td>
<td>2</td>
</tr>
<tr>
<td>Tam Lassiter</td>
<td>Giants</td>
<td>2015</td>
<td>18000000</td>
<td>3</td>
</tr>
<tr>
<td>Buddy Painter</td>
<td>Giants</td>
<td>2015</td>
<td>17277777</td>
<td>4</td>
</tr>
<tr>
<td>Thomas Hillman</td>
<td>Giants</td>
<td>2015</td>
<td>12000000</td>
<td>5</td>
</tr>
<tr>
<td>Alex Paramour</td>
<td>Giants</td>
<td>2015</td>
<td>10250000</td>
<td>6</td>
</tr>
<tr>
<td>Jack Peepers</td>
<td>Giants</td>
<td>2015</td>
<td>9000000</td>
<td>7</td>
</tr>
<tr>
<td>Mark Briste</td>
<td>Giants</td>
<td>2015</td>
<td>8000000</td>
<td>8</td>
</tr>
<tr>
<td>Marcus Bamburger</td>
<td>Giants</td>
<td>2015</td>
<td>6950000</td>
<td>9</td>
</tr>
<tr>
<td>Jalen Ardson</td>
<td>Giants</td>
<td>2015</td>
<td>6000000</td>
<td>10</td>
</tr>
<tr>
<td>Steve Raster</td>
<td>Giants</td>
<td>2015</td>
<td>6000000</td>
<td>10</td>
</tr>
<tr>
<td>Sam Castleman</td>
<td>Giants</td>
<td>2015</td>
<td>5000000</td>
<td>11</td>
</tr>
<tr>
<td>Craig McGawn</td>
<td>Giants</td>
<td>2015</td>
<td>4800000</td>
<td>12</td>
</tr>
<tr>
<td>Norman Aikman</td>
<td>Giants</td>
<td>2015</td>
<td>4000000</td>
<td>13</td>
</tr>
<tr>
<td>Randy Varner</td>
<td>Giants</td>
<td>2015</td>
<td>4000000</td>
<td>13</td>
</tr>
<tr>
<td>Jason Lilliput</td>
<td>Giants</td>
<td>2015</td>
<td>4000000</td>
<td>13</td>
</tr>
<tr>
<td>Billy Bopper</td>
<td>Giants</td>
<td>2015</td>
<td>3600000</td>
<td>14</td>
</tr>
<tr>
<td>Greg Brown</td>
<td>Giants</td>
<td>2015</td>
<td>3600000</td>
<td>14</td>
</tr>
<tr>
<td>Mitch Lovell</td>
<td>Giants</td>
<td>2015</td>
<td>3578825</td>
<td>15</td>
</tr>
<tr>
<td>Bob Cranker</td>
<td>Giants</td>
<td>2015</td>
<td>3175000</td>
<td>16</td>
</tr>
<tr>
<td>Yuri Piamam</td>
<td>Giants</td>
<td>2015</td>
<td>2100000</td>
<td>17</td>
</tr>
<tr>
<td>Joseph Arkman</td>
<td>Giants</td>
<td>2015</td>
<td>1450000</td>
<td>18</td>
</tr>
<tr>
<td>Trevor Imhof</td>
<td>Giants</td>
<td>2015</td>
<td>1100000</td>
<td>19</td>
</tr>
<tr>
<td>Jason Minman</td>
<td>Giants</td>
<td>2015</td>
<td>1000000</td>
<td>20</td>
</tr>
</tbody>
</table>

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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>TEAM</th>
<th>SEASON</th>
<th>SALARY</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mitch Hassleman</td>
<td>Cards</td>
<td>2015</td>
<td>17000000</td>
<td>1</td>
</tr>
<tr>
<td>Yuri Milleton</td>
<td>Cards</td>
<td>2015</td>
<td>15200000</td>
<td>2</td>
</tr>
<tr>
<td>James Grassser</td>
<td>Cards</td>
<td>2015</td>
<td>9375000</td>
<td>3</td>
</tr>
<tr>
<td>Jack Hellman</td>
<td>Cards</td>
<td>2015</td>
<td>8300000</td>
<td>4</td>
</tr>
<tr>
<td>Larry Lintos</td>
<td>Cards</td>
<td>2015</td>
<td>7000000</td>
<td>5</td>
</tr>
<tr>
<td>Jeremy Johnson</td>
<td>Cards</td>
<td>2015</td>
<td>4125000</td>
<td>6</td>
</tr>
<tr>
<td>Mitch Canepa</td>
<td>Cards</td>
<td>2015</td>
<td>3750000</td>
<td>7</td>
</tr>
<tr>
<td>Mitch Brandon</td>
<td>Cards</td>
<td>2015</td>
<td>3500000</td>
<td>8</td>
</tr>
<tr>
<td>Robert Cohen</td>
<td>Cards</td>
<td>2015</td>
<td>3000000</td>
<td>9</td>
</tr>
<tr>
<td>James Woegren</td>
<td>Cards</td>
<td>2015</td>
<td>2675000</td>
<td>10</td>
</tr>
<tr>
<td>Sam Culligan</td>
<td>Cards</td>
<td>2015</td>
<td>2652732</td>
<td>11</td>
</tr>
<tr>
<td>Barry Morse</td>
<td>Cards</td>
<td>2015</td>
<td>2379781</td>
<td>12</td>
</tr>
<tr>
<td>Michael Rastono</td>
<td>Cards</td>
<td>2015</td>
<td>2000000</td>
<td>13</td>
</tr>
<tr>
<td>Carl Vanamos</td>
<td>Cards</td>
<td>2015</td>
<td>2000000</td>
<td>13</td>
</tr>
<tr>
<td>Alex Wister</td>
<td>Cards</td>
<td>2015</td>
<td>1950000</td>
<td>15</td>
</tr>
<tr>
<td>Pablo Bonjourno</td>
<td>Cards</td>
<td>2015</td>
<td>1650000</td>
<td>16</td>
</tr>
<tr>
<td>Jonathan Pearlman</td>
<td>Cards</td>
<td>2015</td>
<td>1500000</td>
<td>17</td>
</tr>
<tr>
<td>Jan Bromley</td>
<td>Cards</td>
<td>2015</td>
<td>1200000</td>
<td>18</td>
</tr>
<tr>
<td>Martin Cassman</td>
<td>Giants</td>
<td>2015</td>
<td>20833333</td>
<td>1</td>
</tr>
<tr>
<td>Harry Pennello</td>
<td>Giants</td>
<td>2015</td>
<td>18500000</td>
<td>2</td>
</tr>
<tr>
<td>Tam Lassiter</td>
<td>Giants</td>
<td>2015</td>
<td>18000000</td>
<td>3</td>
</tr>
<tr>
<td>Buddy Painter</td>
<td>Giants</td>
<td>2015</td>
<td>17277777</td>
<td>4</td>
</tr>
<tr>
<td>Thomas Hillman</td>
<td>Giants</td>
<td>2015</td>
<td>12000000</td>
<td>5</td>
</tr>
<tr>
<td>Alex Paramour</td>
<td>Giants</td>
<td>2015</td>
<td>10250000</td>
<td>6</td>
</tr>
<tr>
<td>Jack Peepers</td>
<td>Giants</td>
<td>2015</td>
<td>9000000</td>
<td>7</td>
</tr>
<tr>
<td>Mark Briste</td>
<td>Giants</td>
<td>2015</td>
<td>8000000</td>
<td>8</td>
</tr>
<tr>
<td>Marcus Bamburger</td>
<td>Giants</td>
<td>2015</td>
<td>6950000</td>
<td>9</td>
</tr>
<tr>
<td>Jalen Ardson</td>
<td>Giants</td>
<td>2015</td>
<td>6000000</td>
<td>10</td>
</tr>
<tr>
<td>Steve Raster</td>
<td>Giants</td>
<td>2015</td>
<td>6000000</td>
<td>10</td>
</tr>
<tr>
<td>Sam Castleman</td>
<td>Giants</td>
<td>2015</td>
<td>5000000</td>
<td>12</td>
</tr>
<tr>
<td>Craig McGawn</td>
<td>Giants</td>
<td>2015</td>
<td>4800000</td>
<td>13</td>
</tr>
<tr>
<td>Norman Aikman</td>
<td>Giants</td>
<td>2015</td>
<td>4000000</td>
<td>14</td>
</tr>
<tr>
<td>Randy Varner</td>
<td>Giants</td>
<td>2015</td>
<td>4000000</td>
<td>14</td>
</tr>
<tr>
<td>Jason Lilliput</td>
<td>Giants</td>
<td>2015</td>
<td>4000000</td>
<td>14</td>
</tr>
<tr>
<td>Billy Bopper</td>
<td>Giants</td>
<td>2015</td>
<td>3600000</td>
<td>17</td>
</tr>
<tr>
<td>Greg Brown</td>
<td>Giants</td>
<td>2015</td>
<td>3600000</td>
<td>17</td>
</tr>
<tr>
<td>Mitch Lovell</td>
<td>Giants</td>
<td>2015</td>
<td>3578825</td>
<td>19</td>
</tr>
<tr>
<td>Bob Cranker</td>
<td>Giants</td>
<td>2015</td>
<td>3175000</td>
<td>20</td>
</tr>
<tr>
<td>Yuri Piamam</td>
<td>Giants</td>
<td>2015</td>
<td>2100000</td>
<td>21</td>
</tr>
<tr>
<td>Joseph Arkman</td>
<td>Giants</td>
<td>2015</td>
<td>1450000</td>
<td>22</td>
</tr>
<tr>
<td>Trevor Imhof</td>
<td>Giants</td>
<td>2015</td>
<td>1100000</td>
<td>23</td>
</tr>
<tr>
<td>Jason Minman</td>
<td>Giants</td>
<td>2015</td>
<td>1000000</td>
<td>24</td>
</tr>
</tbody>
</table>

42 rows selected
See Also

» Window and Aggregate functions

» BIGINT data type

» RANK function

» OVER clause

» Using Window Functions in the Developer Guide.
The **DIGITS** function returns a fixed-length character string representation of a numeric value.

**Syntax**

```sql
DIGITS( val )
```

*val*

The value that you want converted into a string representation. This can be a numeric, CHAR, or VARCHAR value.

**Results**

This function converts the input `val` into a fixed-length string.

The length of the output string is determined by the data type of the `val` argument, as shown in the following table:

<table>
<thead>
<tr>
<th>Input Value Type</th>
<th>Length of Result String</th>
</tr>
</thead>
<tbody>
<tr>
<td>TINYINT, SMALLINT</td>
<td>5</td>
</tr>
<tr>
<td>INT</td>
<td>10</td>
</tr>
<tr>
<td>BIGINT</td>
<td>19</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>The precision of the DECIMAL value.</td>
</tr>
<tr>
<td>FLOAT with precision &lt;= 23, REAL</td>
<td>23</td>
</tr>
<tr>
<td>FLOAT with precision &gt; 23, DOUBLE</td>
<td>52</td>
</tr>
<tr>
<td>CHAR, VARCHAR</td>
<td>31</td>
</tr>
</tbody>
</table>

(The value is converted to DECIMAL(31,6))

The return value is left-padded with 0’s if the number of digits is less than the fixed-length of the result string.
Examples

splice> VALUES( DIGITS( 12345 ) );
1
--------------
0000012345

splice> VALUES( DIGITS( 123.45 ) );
1
--------------
12345

splice> VALUES( DIGITS( '12345' ) );
1
------------------------
000000000000000001234500000

splice> VALUES( DIGITS( '123.45' ) );
1
------------------------
000000000000000001234500000
The **DOUBLE** function returns a floating-point number corresponding to a:

- a number if the argument is a numeric expression
- a 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
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";

<table>
<thead>
<tr>
<th>BIRTHDATE</th>
<th>Quarter</th>
<th>Week</th>
<th>Weekday</th>
</tr>
</thead>
<tbody>
<tr>
<td>1987-03-27</td>
<td>1</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>1987-01-21</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>1991-01-15</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1982-01-05</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1990-03-22</td>
<td>1</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>1989-01-01</td>
<td>1</td>
<td>52</td>
<td>7</td>
</tr>
<tr>
<td>1988-04-20</td>
<td>2</td>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>1983-04-13</td>
<td>2</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>1990-06-16</td>
<td>2</td>
<td>24</td>
<td>6</td>
</tr>
<tr>
<td>1984-04-11</td>
<td>2</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>1981-07-02</td>
<td>3</td>
<td>27</td>
<td>4</td>
</tr>
<tr>
<td>1977-08-30</td>
<td>3</td>
<td>35</td>
<td>2</td>
</tr>
<tr>
<td>1989-08-21</td>
<td>3</td>
<td>34</td>
<td>1</td>
</tr>
<tr>
<td>1984-09-21</td>
<td>3</td>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>1990-10-30</td>
<td>4</td>
<td>44</td>
<td>2</td>
</tr>
<tr>
<td>1983-12-24</td>
<td>4</td>
<td>51</td>
<td>6</td>
</tr>
<tr>
<td>1983-11-06</td>
<td>4</td>
<td>44</td>
<td>7</td>
</tr>
<tr>
<td>1982-10-12</td>
<td>4</td>
<td>41</td>
<td>2</td>
</tr>
<tr>
<td>1989-11-17</td>
<td>4</td>
<td>46</td>
<td>5</td>
</tr>
</tbody>
</table>

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
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 date 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:

```sql
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;
```

<table>
<thead>
<tr>
<th>TEAM</th>
<th>DISPLAYNAME</th>
<th>HOMER&amp;</th>
<th>Least</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cards</td>
<td>Mitch Canepa</td>
<td>28</td>
<td>11</td>
</tr>
<tr>
<td>Cards</td>
<td>Jonathan Pearlman</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>Cards</td>
<td>Roger Green</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>Cards</td>
<td>Michael Rastono</td>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>Cards</td>
<td>Jack Hellman</td>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>Cards</td>
<td>Kelly Wacherman</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Giants</td>
<td>Bob Cranker</td>
<td>21</td>
<td>12</td>
</tr>
<tr>
<td>Giants</td>
<td>Buddy Painter</td>
<td>19</td>
<td>12</td>
</tr>
<tr>
<td>Giants</td>
<td>Billy Bopper</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>Giants</td>
<td>Mitch Duffer</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

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
GROUP_USER

When used outside stored routines, GROUP_USER returns the names of the groups to which the current user belongs.

If used within a stored routine created with EXTERNAL SECURITY DEFINER, however, GROUP_USER returns the groups to which the user who owns the schema of the routine belongs; this is usually the creating user, although the database owner could be the creator as well.

This function returns a string of up to 32672 characters.

Syntax

GROUP_USER

Example

splice> VALUES GROUP_USER;
1
--------------------------------------------------------------------
"VIEWERS", "EDITORS"
1 row selected

See Also

» USER function
» SESSION_USER function
» CURRENT_USER function
» CREATE_FUNCTION statement
» CREATE_PROCEDURE statement
HOUR

The HOUR function returns the hour part of a value.

Syntax

```sql
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 NULL value.

Example

```sql
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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex Paramour</td>
<td>6</td>
</tr>
<tr>
<td>Buddy Painter</td>
<td>7</td>
</tr>
<tr>
<td>Jeremy Packman</td>
<td>8</td>
</tr>
<tr>
<td>Pablo Bonjourno</td>
<td>1</td>
</tr>
<tr>
<td>Paul Kaster</td>
<td>1</td>
</tr>
</tbody>
</table>

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[GER] (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 NULL value.
- 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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>Innings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marcus Bamburger</td>
<td>218</td>
</tr>
<tr>
<td>Jason Larrimore</td>
<td>218</td>
</tr>
<tr>
<td>Milt Warrimore</td>
<td>181</td>
</tr>
<tr>
<td>Carl Marin</td>
<td>179</td>
</tr>
<tr>
<td>Charles Heillman</td>
<td>177</td>
</tr>
<tr>
<td>Larry Lintos</td>
<td>175</td>
</tr>
<tr>
<td>Randy Varner</td>
<td>135</td>
</tr>
<tr>
<td>James Grasser</td>
<td>129</td>
</tr>
<tr>
<td>Thomas Hillman</td>
<td>123</td>
</tr>
<tr>
<td>Jack Peepers</td>
<td>110</td>
</tr>
<tr>
<td>Tam Lassiter</td>
<td>76</td>
</tr>
<tr>
<td>Yuri Piamam</td>
<td>76</td>
</tr>
<tr>
<td>Ken Straiter</td>
<td>74</td>
</tr>
<tr>
<td>Gary Kosovo</td>
<td>73</td>
</tr>
<tr>
<td>Tom Rather</td>
<td>68</td>
</tr>
<tr>
<td>Steve Mossely</td>
<td>63</td>
</tr>
<tr>
<td>Carl Vanamos</td>
<td>61</td>
</tr>
<tr>
<td>Martin Cassman</td>
<td>60</td>
</tr>
<tr>
<td>Tim Lentleson</td>
<td>60</td>
</tr>
<tr>
<td>Sam Castleman</td>
<td>58</td>
</tr>
<tr>
<td>Steve Raster</td>
<td>57</td>
</tr>
<tr>
<td>Mitch Lovell</td>
<td>55</td>
</tr>
<tr>
<td>Harold Sermer</td>
<td>51</td>
</tr>
</tbody>
</table>

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;

<table>
<thead>
<tr>
<th>POS &amp; ID</th>
<th>SALARY</th>
<th>PrevHigherSalary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1B</td>
<td>2</td>
<td>3600000</td>
</tr>
<tr>
<td>1B</td>
<td>63</td>
<td>2379781</td>
</tr>
<tr>
<td>1B</td>
<td>50</td>
<td>2000000</td>
</tr>
<tr>
<td>3B</td>
<td>14</td>
<td>4800000</td>
</tr>
<tr>
<td>3B</td>
<td>53</td>
<td>3750000</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>17277777</td>
</tr>
<tr>
<td>C</td>
<td>49</td>
<td>15200000</td>
</tr>
<tr>
<td>CF</td>
<td>7</td>
<td>10250000</td>
</tr>
<tr>
<td>CF</td>
<td>59</td>
<td>4125000</td>
</tr>
<tr>
<td>CF</td>
<td>55</td>
<td>1650000</td>
</tr>
<tr>
<td>LF</td>
<td>54</td>
<td>17000000</td>
</tr>
<tr>
<td>LF</td>
<td>6</td>
<td>4000000</td>
</tr>
<tr>
<td>LF</td>
<td>27</td>
<td>1100000</td>
</tr>
<tr>
<td>P</td>
<td>34</td>
<td>20833333</td>
</tr>
<tr>
<td>P</td>
<td>33</td>
<td>18000000</td>
</tr>
<tr>
<td>P</td>
<td>31</td>
<td>12000000</td>
</tr>
<tr>
<td>P</td>
<td>76</td>
<td>9375000</td>
</tr>
<tr>
<td>P</td>
<td>32</td>
<td>9000000</td>
</tr>
<tr>
<td>P</td>
<td>75</td>
<td>7000000</td>
</tr>
<tr>
<td>P</td>
<td>28</td>
<td>6950000</td>
</tr>
<tr>
<td>P</td>
<td>40</td>
<td>6000000</td>
</tr>
<tr>
<td>P</td>
<td>41</td>
<td>6000000</td>
</tr>
<tr>
<td>P</td>
<td>46</td>
<td>5000000</td>
</tr>
<tr>
<td>P</td>
<td>30</td>
<td>4000000</td>
</tr>
<tr>
<td>P</td>
<td>43</td>
<td>4000000</td>
</tr>
<tr>
<td>P</td>
<td>35</td>
<td>3578825</td>
</tr>
<tr>
<td>P</td>
<td>86</td>
<td>3500000</td>
</tr>
<tr>
<td>P</td>
<td>82</td>
<td>3000000</td>
</tr>
<tr>
<td>P</td>
<td>88</td>
<td>2675000</td>
</tr>
<tr>
<td>P</td>
<td>90</td>
<td>2652732</td>
</tr>
<tr>
<td>P</td>
<td>36</td>
<td>2100000</td>
</tr>
<tr>
<td>P</td>
<td>79</td>
<td>2000000</td>
</tr>
<tr>
<td>P</td>
<td>80</td>
<td>1950000</td>
</tr>
<tr>
<td>P</td>
<td>94</td>
<td>1200000</td>
</tr>
<tr>
<td>RF</td>
<td>8</td>
<td>18500000</td>
</tr>
<tr>
<td>RF</td>
<td>56</td>
<td>8300000</td>
</tr>
<tr>
<td>RF</td>
<td>12</td>
<td>8000000</td>
</tr>
<tr>
<td>RF</td>
<td>10</td>
<td>1000000</td>
</tr>
<tr>
<td>SS</td>
<td>4</td>
<td>3175000</td>
</tr>
<tr>
<td>SS</td>
<td>52</td>
<td>1500000</td>
</tr>
<tr>
<td>UT</td>
<td>17</td>
<td>1450000</td>
</tr>
</tbody>
</table>

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);

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>BirthDate</th>
<th>MonthEnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tam Croonster</td>
<td>1980-12-19</td>
<td>1980-12-31</td>
</tr>
<tr>
<td>Jason Martell</td>
<td>1982-02-01</td>
<td>1982-02-28</td>
</tr>
<tr>
<td>Kameron Fannais</td>
<td>1982-05-24</td>
<td>1982-05-31</td>
</tr>
<tr>
<td>Greg Brown</td>
<td>1983-12-24</td>
<td>1983-12-31</td>
</tr>
<tr>
<td>Edward Erdman</td>
<td>1985-12-21</td>
<td>1985-12-31</td>
</tr>
<tr>
<td>Reed Lister</td>
<td>1986-12-16</td>
<td>1986-12-31</td>
</tr>
<tr>
<td>Larry Lintos</td>
<td>1987-05-12</td>
<td>1987-05-31</td>
</tr>
<tr>
<td>Taylor Trantula</td>
<td>1987-12-17</td>
<td>1987-12-31</td>
</tr>
<tr>
<td>Tim Lentleson</td>
<td>1988-02-21</td>
<td>1988-02-29</td>
</tr>
<tr>
<td>Cameron Silliman</td>
<td>1988-12-21</td>
<td>1988-12-31</td>
</tr>
<tr>
<td>Tom Rather</td>
<td>1990-05-29</td>
<td>1990-05-31</td>
</tr>
<tr>
<td>Mo Grandosi</td>
<td>1992-02-16</td>
<td>1992-02-29</td>
</tr>
</tbody>
</table>

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:

```sql
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;
```

<table>
<thead>
<tr>
<th>TEAM</th>
<th>DISPLAYNAME</th>
<th>HOMER&amp;</th>
<th>Most</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cards</td>
<td>Mitch Canepa</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>Cards</td>
<td>Jonathan Pearlman</td>
<td>17</td>
<td>28</td>
</tr>
<tr>
<td>Cards</td>
<td>Roger Green</td>
<td>17</td>
<td>28</td>
</tr>
<tr>
<td>Cards</td>
<td>Michael Rastono</td>
<td>13</td>
<td>28</td>
</tr>
<tr>
<td>Cards</td>
<td>Jack Hellman</td>
<td>13</td>
<td>28</td>
</tr>
<tr>
<td>Cards</td>
<td>Kelly Wacherman</td>
<td>11</td>
<td>28</td>
</tr>
<tr>
<td>Giants</td>
<td>Bob Cranker</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>Giants</td>
<td>Buddy Painter</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>Giants</td>
<td>Billy Bopper</td>
<td>18</td>
<td>21</td>
</tr>
<tr>
<td>Giants</td>
<td>Mitch Duffer</td>
<td>12</td>
<td>21</td>
</tr>
</tbody>
</table>

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 unless you specify the Locale parameter.

**Syntax**

```
LCASE or LOWER ( CharacterExpression [, Locale ] )
```

*CharacterExpression*

A LONG VARCHAR data type, or any built-in type that is implicitly converted to a string (but not a bit expression).

*Locale*

Optional. You can specify the locale (language and country) to perform language-specific conversions. If you don't supply this parameter, the Java Session locale value, which is automatically detected, is used as your locale.

You must specify the locale using the following format: ll_CC, where:


For example, you can use de_DE to specify that German language rules should be used when applying this function, or you could specify en_US to apply American English rules.

**Results**

If the CharacterExpression evaluates to NULL, this function returns NULL.

In general, the type, length, and maximum length of the returned value are the same as the length and maximum length of the CharacterExpression. However, the data type, length, and maximum length of the result can be different if you're using a locale value that differs from the default locale of your database.
This is because a single character may convert into multiple characters, when a location value is involved. For example, if you're applying this function to a CHAR value and the resulting value length exceeds the limits of a CHAR value, the result will be a VARCHAR value. Similarly, converting a VARCHAR value may result in a LONG VARCHAR value, and converting a LONG VARCHAR value may result in a CLOB value.

**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

splice> SELECT LOWER( DisplayName, 'en_US' ) 
    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 date 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;

<table>
<thead>
<tr>
<th>DisplayName</th>
<th>POS&amp;</th>
<th>SALARY</th>
<th>NextLowerSalary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Billy Bopper</td>
<td>1B</td>
<td>3600000</td>
<td>2379781</td>
</tr>
<tr>
<td>Barry Morse</td>
<td>1B</td>
<td>237978</td>
<td>2000000</td>
</tr>
<tr>
<td>Michael Rastono</td>
<td>1B</td>
<td>2000000</td>
<td>NULL</td>
</tr>
<tr>
<td>Craig McGawn</td>
<td>3B</td>
<td>4800000</td>
<td>3750000</td>
</tr>
<tr>
<td>Mitch Canepa</td>
<td>3B</td>
<td>3750000</td>
<td>NULL</td>
</tr>
<tr>
<td>Buddy Painter</td>
<td>C</td>
<td>17277777</td>
<td>15200000</td>
</tr>
<tr>
<td>Yuri Milleton</td>
<td>C</td>
<td>15200000</td>
<td>NULL</td>
</tr>
<tr>
<td>Alex Paramour</td>
<td>CF</td>
<td>10250000</td>
<td>4125000</td>
</tr>
<tr>
<td>Jeremy Johnson</td>
<td>CF</td>
<td>4125000</td>
<td>1650000</td>
</tr>
<tr>
<td>Pablo Bonjournio</td>
<td>CF</td>
<td>1650000</td>
<td>NULL</td>
</tr>
<tr>
<td>Mitch Hassleman</td>
<td>LF</td>
<td>17000000</td>
<td>4000000</td>
</tr>
<tr>
<td>Norman Aikman</td>
<td>LF</td>
<td>4000000</td>
<td>1100000</td>
</tr>
<tr>
<td>Trevor Imhof</td>
<td>LF</td>
<td>1100000</td>
<td>NULL</td>
</tr>
<tr>
<td>Greg Brown</td>
<td>OF</td>
<td>3600000</td>
<td>NULL</td>
</tr>
<tr>
<td>Martin Cassman</td>
<td>P</td>
<td>20833333</td>
<td>18000000</td>
</tr>
<tr>
<td>Tam Lassiter</td>
<td>P</td>
<td>18000000</td>
<td>12000000</td>
</tr>
<tr>
<td>Thomas Hillman</td>
<td>P</td>
<td>12000000</td>
<td>9375000</td>
</tr>
<tr>
<td>James Grasser</td>
<td>P</td>
<td>9375000</td>
<td>9000000</td>
</tr>
<tr>
<td>Jack Peepers</td>
<td>P</td>
<td>9000000</td>
<td>7000000</td>
</tr>
<tr>
<td>Larry Lintos</td>
<td>P</td>
<td>7000000</td>
<td>6950000</td>
</tr>
<tr>
<td>Marcus Bamburger</td>
<td>P</td>
<td>6950000</td>
<td>6000000</td>
</tr>
<tr>
<td>Jalen Ardson</td>
<td>P</td>
<td>6000000</td>
<td>6000000</td>
</tr>
<tr>
<td>Steve Raster</td>
<td>P</td>
<td>6000000</td>
<td>5000000</td>
</tr>
<tr>
<td>Sam Castleman</td>
<td>P</td>
<td>5000000</td>
<td>4000000</td>
</tr>
<tr>
<td>Randy Varner</td>
<td>P</td>
<td>4000000</td>
<td>4000000</td>
</tr>
<tr>
<td>Jason Lilliput</td>
<td>P</td>
<td>4000000</td>
<td>3578825</td>
</tr>
<tr>
<td>Mitch Lovell</td>
<td>P</td>
<td>3578825</td>
<td>3500000</td>
</tr>
<tr>
<td>Mitch Brandon</td>
<td>P</td>
<td>3500000</td>
<td>3000000</td>
</tr>
<tr>
<td>Robert Cohen</td>
<td>P</td>
<td>3000000</td>
<td>2675000</td>
</tr>
<tr>
<td>James Woegren</td>
<td>P</td>
<td>2675000</td>
<td>2652732</td>
</tr>
<tr>
<td>Sam Culligan</td>
<td>P</td>
<td>2652732</td>
<td>2100000</td>
</tr>
<tr>
<td>Yuri Piamam</td>
<td>P</td>
<td>2100000</td>
<td>2000000</td>
</tr>
<tr>
<td>Carl Vanamos</td>
<td>P</td>
<td>2000000</td>
<td>1950000</td>
</tr>
<tr>
<td>Alex Wister</td>
<td>P</td>
<td>1950000</td>
<td>1200000</td>
</tr>
<tr>
<td>Jan Bromley</td>
<td>P</td>
<td>1200000</td>
<td>NULL</td>
</tr>
<tr>
<td>Harry Pennello</td>
<td>RF</td>
<td>18500000</td>
<td>8300000</td>
</tr>
<tr>
<td>Jack Hellman</td>
<td>RF</td>
<td>8300000</td>
<td>8000000</td>
</tr>
<tr>
<td>Mark Briste</td>
<td>RF</td>
<td>8000000</td>
<td>1000000</td>
</tr>
<tr>
<td>Jason Minman</td>
<td>RF</td>
<td>1000000</td>
<td>NULL</td>
</tr>
<tr>
<td>Bob Cranker</td>
<td>SS</td>
<td>3175000</td>
<td>1500000</td>
</tr>
<tr>
<td>Jonathan Pearlman</td>
<td>SS</td>
<td>1500000</td>
<td>NULL</td>
</tr>
<tr>
<td>Joseph Arkman</td>
<td>UT</td>
<td>1450000</td>
<td>NULL</td>
</tr>
</tbody>
</table>

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.
**LEFT**

The **LEFT** function returns the leftmost n characters from a string.

**Syntax**

```
LEFT( string, n )
```

*string*  
The string from which you want to extract characters.

*n*  
An integer value that specifies the number of characters that you want to extract from *string*.

**Results**

This function takes as arguments a string from which you want to extract a substring, starting with the first (leftmost) character in the string, and an integer that specifies the number of characters that you want to extract.

If there are fewer characters in the *string* than are specified in *n*, the entire *string* is returned (no padding is applied to the result).

**Examples**

```
splice> VALUES( LEFT( 'Splice Machine', 6 ) );
1 ------------
Splice

splice> VALUES( LEFT( 'Splice Machine', 3 ) );
1 ------------
Spl

splice> VALUES( LEFT( 'Splice Machine', 32 ) );
1 ------------------------
Splice Machine
```
**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";

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>NameLen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greg Brown</td>
<td>10</td>
</tr>
<tr>
<td>John Purser</td>
<td>11</td>
</tr>
<tr>
<td>Bob Cranker</td>
<td>11</td>
</tr>
<tr>
<td>Billy Bopper</td>
<td>12</td>
</tr>
<tr>
<td>Mitch Duffer</td>
<td>12</td>
</tr>
<tr>
<td>Jason Minman</td>
<td>12</td>
</tr>
<tr>
<td>Buddy Painter</td>
<td>13</td>
</tr>
<tr>
<td>Norman Aikman</td>
<td>13</td>
</tr>
<tr>
<td>Alex Paramour</td>
<td>13</td>
</tr>
<tr>
<td>Harry Pennello</td>
<td>14</td>
</tr>
</tbody>
</table>

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;

<table>
<thead>
<tr>
<th>ID</th>
<th>SMALLINT</th>
<th>INT</th>
<th>BIGINT</th>
<th>DECIMAL5</th>
<th>DECIMAL15</th>
<th>DECIMAL30</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>16</td>
</tr>
</tbody>
</table>

10 rows selected

splice> VALUES LENGTH(X'FF'),
    LENGTH(X'FFFF'),
    LENGTH(X'FFFFFFFF'),
    LENGTH(X'FFFFFFFFFFFFFFFF');

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
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**

```sql
LN ( number )
LOG ( number )
```

*number*

A **DOUBLE PRECISION** number that is greater than zero (0).

**Example**

```sql
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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex Paramour</td>
<td>6</td>
</tr>
<tr>
<td>Buddy Painter</td>
<td>7</td>
</tr>
<tr>
<td>Jeremy Packman</td>
<td>8</td>
</tr>
<tr>
<td>Pablo Bonjourno</td>
<td>0</td>
</tr>
<tr>
<td>Paul Kaster</td>
<td>0</td>
</tr>
</tbody>
</table>

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 an window (analytic) function.

**Syntax**

\[
\text{MAX ( [ DISTINCT | ALL ] Expression )}
\]

**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:

```sql
--- 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:

```sql
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:

```sql
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;

<table>
<thead>
<tr>
<th>TEAM</th>
<th>DISPLAYNAME</th>
<th>HOME_RUNS</th>
<th>MOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cards</td>
<td>Mitch Canepa</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>Cards</td>
<td>Jonathan Pearlman</td>
<td>17</td>
<td>28</td>
</tr>
<tr>
<td>Cards</td>
<td>Roger Green</td>
<td>17</td>
<td>28</td>
</tr>
<tr>
<td>Cards</td>
<td>Michael Rastono</td>
<td>13</td>
<td>28</td>
</tr>
<tr>
<td>Cards</td>
<td>Jack Hellman</td>
<td>13</td>
<td>28</td>
</tr>
<tr>
<td>Cards</td>
<td>Kelly Wacherman</td>
<td>11</td>
<td>28</td>
</tr>
<tr>
<td>Giants</td>
<td>Bob Cranker</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>Giants</td>
<td>Buddy Painter</td>
<td>19</td>
<td>21</td>
</tr>
<tr>
<td>Giants</td>
<td>Billy Bopper</td>
<td>18</td>
<td>21</td>
</tr>
<tr>
<td>Giants</td>
<td>Mitch Duffer</td>
<td>12</td>
<td>21</td>
</tr>
</tbody>
</table>

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**

```sql
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:

```sql
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 NULL value.

Example

splice> VALUES( NOW, HOUR(NOW), MINUTE(NOW), SECOND(NOW) );

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>48</td>
<td>55.217</td>
</tr>
</tbody>
</table>

1 row selected

See Also

» About Data Types
» TIMESTAMP data value
» HOUR function
» SECOND function
» TIMESTAMP function
» TIMESTAMDIFF function
MOD

MOD returns the remainder (modulus) of argument 1 divided by argument 2.

Syntax

\[
\text{mod}(\text{number}, \text{divisor})
\]

- **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.

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;

<table>
<thead>
<tr>
<th>ID</th>
<th>TEAM</th>
<th>DISPLAYNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Giants</td>
<td>Buddy Painter</td>
</tr>
<tr>
<td>3</td>
<td>Giants</td>
<td>John Purser</td>
</tr>
<tr>
<td>5</td>
<td>Giants</td>
<td>Mitch Duffer</td>
</tr>
<tr>
<td>7</td>
<td>Giants</td>
<td>Alex Paramour</td>
</tr>
<tr>
<td>9</td>
<td>Giants</td>
<td>Greg Brown</td>
</tr>
<tr>
<td>11</td>
<td>Giants</td>
<td>Kelly Tamlin</td>
</tr>
<tr>
<td>13</td>
<td>Giants</td>
<td>Andy Sussman</td>
</tr>
<tr>
<td>15</td>
<td>Giants</td>
<td>Elliot Andrews</td>
</tr>
<tr>
<td>17</td>
<td>Giants</td>
<td>Joseph Arkman</td>
</tr>
<tr>
<td>19</td>
<td>Giants</td>
<td>Jeremy Packman</td>
</tr>
<tr>
<td>21</td>
<td>Giants</td>
<td>Jason Pratter</td>
</tr>
<tr>
<td>23</td>
<td>Giants</td>
<td>Nathan Nickels</td>
</tr>
<tr>
<td>25</td>
<td>Giants</td>
<td>Reed Lister</td>
</tr>
<tr>
<td>27</td>
<td>Giants</td>
<td>Trevor Imhof</td>
</tr>
<tr>
<td>29</td>
<td>Giants</td>
<td>Charles Heillman</td>
</tr>
<tr>
<td>31</td>
<td>Giants</td>
<td>Thomas Hillman</td>
</tr>
<tr>
<td>33</td>
<td>Giants</td>
<td>Tam Lassiter</td>
</tr>
<tr>
<td>35</td>
<td>Giants</td>
<td>Mitch Lovell</td>
</tr>
<tr>
<td>37</td>
<td>Giants</td>
<td>Justin Oscar</td>
</tr>
<tr>
<td>39</td>
<td>Giants</td>
<td>Gary Kosovo</td>
</tr>
<tr>
<td>41</td>
<td>Giants</td>
<td>Steve Raster</td>
</tr>
<tr>
<td>43</td>
<td>Giants</td>
<td>Jason Lilliput</td>
</tr>
<tr>
<td>45</td>
<td>Giants</td>
<td>Cory Hammersmith</td>
</tr>
<tr>
<td>47</td>
<td>Giants</td>
<td>Barry Bochner</td>
</tr>
<tr>
<td>49</td>
<td>Cards</td>
<td>Yuri Milleton</td>
</tr>
<tr>
<td>51</td>
<td>Cards</td>
<td>Kelly Wacherman</td>
</tr>
<tr>
<td>53</td>
<td>Cards</td>
<td>Mitch Canepa</td>
</tr>
<tr>
<td>55</td>
<td>Cards</td>
<td>Pablo Bonjourno</td>
</tr>
<tr>
<td>57</td>
<td>Cards</td>
<td>Roger Green</td>
</tr>
<tr>
<td>59</td>
<td>Cards</td>
<td>Jeremy Johnson</td>
</tr>
<tr>
<td>61</td>
<td>Cards</td>
<td>Tad Philomen</td>
</tr>
<tr>
<td>63</td>
<td>Cards</td>
<td>Barry Morse</td>
</tr>
<tr>
<td>65</td>
<td>Cards</td>
<td>George Goomba</td>
</tr>
<tr>
<td>67</td>
<td>Cards</td>
<td>David Janssen</td>
</tr>
<tr>
<td>69</td>
<td>Cards</td>
<td>Edward Erdman</td>
</tr>
<tr>
<td>71</td>
<td>Cards</td>
<td>Don Allison</td>
</tr>
<tr>
<td>73</td>
<td>Cards</td>
<td>Carl Marin</td>
</tr>
<tr>
<td>Cards</td>
<td>Larry Lintos</td>
<td></td>
</tr>
<tr>
<td>Cards</td>
<td>Tim Lentleson</td>
<td></td>
</tr>
<tr>
<td>Cards</td>
<td>Carl Vanamos</td>
<td></td>
</tr>
<tr>
<td>Cards</td>
<td>Steve Mossely</td>
<td></td>
</tr>
<tr>
<td>Cards</td>
<td>Manny Stolanaro</td>
<td></td>
</tr>
<tr>
<td>Cards</td>
<td>Michael Hillson</td>
<td></td>
</tr>
<tr>
<td>Cards</td>
<td>Neil Gaston</td>
<td></td>
</tr>
<tr>
<td>Cards</td>
<td>Mo Grandosi</td>
<td></td>
</tr>
<tr>
<td>Cards</td>
<td>Mark Hasty</td>
<td></td>
</tr>
<tr>
<td>Cards</td>
<td>Stephen Tuvesco</td>
<td></td>
</tr>
</tbody>
</table>

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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>TEAM</th>
<th>BIRTHDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greg Brown</td>
<td>Giants</td>
<td>1983-12-24</td>
</tr>
<tr>
<td>Reed Lister</td>
<td>Giants</td>
<td>1986-12-16</td>
</tr>
<tr>
<td>Cameron Silliman</td>
<td>Cards</td>
<td>1988-12-21</td>
</tr>
<tr>
<td>Edward Erdman</td>
<td>Cards</td>
<td>1985-12-21</td>
</tr>
<tr>
<td>Taylor Trantula</td>
<td>Cards</td>
<td>1987-12-17</td>
</tr>
<tr>
<td>Tam Croonster</td>
<td>Cards</td>
<td>1980-12-19</td>
</tr>
</tbody>
</table>

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);

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob Cranker</td>
<td>January</td>
</tr>
<tr>
<td>Mitch Duffer</td>
<td>January</td>
</tr>
<tr>
<td>Norman Aikman</td>
<td>January</td>
</tr>
<tr>
<td>Jeremy Packman</td>
<td>January</td>
</tr>
<tr>
<td>Buddy Painter</td>
<td>March</td>
</tr>
<tr>
<td>Andy Sussman</td>
<td>March</td>
</tr>
<tr>
<td>Billy Bopper</td>
<td>April</td>
</tr>
<tr>
<td>Harry Pennello</td>
<td>April</td>
</tr>
<tr>
<td>Alex Darba</td>
<td>April</td>
</tr>
<tr>
<td>Kelly Tamlin</td>
<td>June</td>
</tr>
<tr>
<td>Alex Paramour</td>
<td>July</td>
</tr>
<tr>
<td>Mark Briste</td>
<td>August</td>
</tr>
<tr>
<td>Elliot Andrews</td>
<td>August</td>
</tr>
<tr>
<td>Joseph Arkman</td>
<td>September</td>
</tr>
<tr>
<td>John Purser</td>
<td>October</td>
</tr>
<tr>
<td>Craig McGawn</td>
<td>October</td>
</tr>
<tr>
<td>Jason Minman</td>
<td>November</td>
</tr>
<tr>
<td>Henry Socomy</td>
<td>November</td>
</tr>
<tr>
<td>Greg Brown</td>
<td>December</td>
</tr>
</tbody>
</table>

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):

<table>
<thead>
<tr>
<th>Day Name</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunday</td>
<td>Sun</td>
</tr>
<tr>
<td>Monday</td>
<td>Mon</td>
</tr>
<tr>
<td>Tuesday</td>
<td>Tue</td>
</tr>
<tr>
<td>Wednesday</td>
<td>Wed</td>
</tr>
<tr>
<td>Thursday</td>
<td>Thu</td>
</tr>
<tr>
<td>Friday</td>
<td>Fri</td>
</tr>
<tr>
<td>Saturday</td>
<td>Sat</td>
</tr>
</tbody>
</table>

**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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>BIRTHDATE</th>
<th>FirstSunday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buddy Painter</td>
<td>1987-03-27</td>
<td>1987-03-29</td>
</tr>
<tr>
<td>Billy Bopper</td>
<td>1988-04-20</td>
<td>1988-04-24</td>
</tr>
<tr>
<td>John Purser</td>
<td>1990-10-30</td>
<td>1990-11-04</td>
</tr>
<tr>
<td>Bob Cranker</td>
<td>1987-01-21</td>
<td>1987-01-25</td>
</tr>
<tr>
<td>Norman Aikman</td>
<td>1982-01-05</td>
<td>1982-01-10</td>
</tr>
<tr>
<td>Alex Paramour</td>
<td>1981-07-02</td>
<td>1981-07-05</td>
</tr>
<tr>
<td>Harry Pennello</td>
<td>1983-04-13</td>
<td>1983-04-17</td>
</tr>
<tr>
<td>Greg Brown</td>
<td>1983-12-24</td>
<td>1983-12-25</td>
</tr>
<tr>
<td>Jason Minman</td>
<td>1983-11-06</td>
<td>1983-11-06</td>
</tr>
<tr>
<td>Kelly Tamlin</td>
<td>1990-06-16</td>
<td>1990-06-17</td>
</tr>
<tr>
<td>Mark Briste</td>
<td>1977-08-30</td>
<td>1977-09-04</td>
</tr>
<tr>
<td>Andy Sussman</td>
<td>1990-03-22</td>
<td>1990-03-25</td>
</tr>
<tr>
<td>Craig McGawn</td>
<td>1982-10-12</td>
<td>1982-10-17</td>
</tr>
<tr>
<td>Elliot Andrews</td>
<td>1989-08-21</td>
<td>1989-08-27</td>
</tr>
<tr>
<td>Alex Darba</td>
<td>1984-04-11</td>
<td>1984-04-15</td>
</tr>
<tr>
<td>Joseph Arkman</td>
<td>1984-09-21</td>
<td>1984-09-23</td>
</tr>
<tr>
<td>Henry Socomy</td>
<td>1989-11-17</td>
<td>1989-11-19</td>
</tr>
</tbody>
</table>

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.

Splice Machine displays **TIME** and **TIMESTAMP** values using the current time zone for the server to which you are connected. **DATE** values, however, are stored as fixed strings, and are not affected by the time zone of the server to which you are connected.

**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**
- **CURRENT_TIME**
- **DATE type**
- **DATE function**
- **DAY**
- **EXTRACT**
- **LASTDAY**
MONTH
MONTH_BETWEEN
MONTHNAME
NEXTDAY
NOW
QUARTER
TIME type
TIME function
TIMESTAMP type
TIMESTAMP function
TO_CHAR
TO_DATE
WEEK
Working with Dates
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;

<table>
<thead>
<tr>
<th>Position Player</th>
<th>Pos&amp;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barry Morse</td>
<td>1B</td>
</tr>
<tr>
<td>David Janssen</td>
<td>1B</td>
</tr>
<tr>
<td>John Purser</td>
<td>2B</td>
</tr>
<tr>
<td>Kelly Tamlin</td>
<td>2B</td>
</tr>
<tr>
<td>Kelly Wacherman</td>
<td>2B</td>
</tr>
<tr>
<td>Mitch Duffer</td>
<td>3B</td>
</tr>
<tr>
<td>Mitch Canepa</td>
<td>3B</td>
</tr>
<tr>
<td>Buddy Painter</td>
<td>C</td>
</tr>
<tr>
<td>Andy Sussman</td>
<td>C</td>
</tr>
<tr>
<td>Yuri Milleton</td>
<td>C</td>
</tr>
<tr>
<td>Edward Erdman</td>
<td>C</td>
</tr>
<tr>
<td>Alex Paramour</td>
<td>CF</td>
</tr>
<tr>
<td>Pablo Bonjourno</td>
<td>CF</td>
</tr>
<tr>
<td>Jeremy Johnson</td>
<td>CF</td>
</tr>
<tr>
<td>Tad Philomen</td>
<td>CF</td>
</tr>
<tr>
<td>Nathan Nickels</td>
<td>IF</td>
</tr>
<tr>
<td>George Goomba</td>
<td>IF</td>
</tr>
<tr>
<td>Don Allison</td>
<td>IF</td>
</tr>
<tr>
<td>Trevor Imhof</td>
<td>LF</td>
</tr>
<tr>
<td>Elliot Andrews</td>
<td>MI</td>
</tr>
<tr>
<td>Greg Brown</td>
<td>OF</td>
</tr>
<tr>
<td>Jeremy Packman</td>
<td>OF</td>
</tr>
<tr>
<td>Jason Pratter</td>
<td>OF</td>
</tr>
<tr>
<td>Reed Lister</td>
<td>OF</td>
</tr>
<tr>
<td>Roger Green</td>
<td>OF</td>
</tr>
<tr>
<td>Charles Heillman</td>
<td>NULL</td>
</tr>
<tr>
<td>Thomas Hillman</td>
<td>NULL</td>
</tr>
<tr>
<td>Tam Lassiter</td>
<td>NULL</td>
</tr>
<tr>
<td>Mitch Lovell</td>
<td>NULL</td>
</tr>
<tr>
<td>Justin Oscar</td>
<td>NULL</td>
</tr>
<tr>
<td>Gary Kosovo</td>
<td>NULL</td>
</tr>
<tr>
<td>Steve Raster</td>
<td>NULL</td>
</tr>
<tr>
<td>Jason Lilliput</td>
<td>NULL</td>
</tr>
<tr>
<td>Cory Hammersmith</td>
<td>NULL</td>
</tr>
<tr>
<td>Barry Bochner</td>
<td>NULL</td>
</tr>
<tr>
<td>Carl Marin</td>
<td>NULL</td>
</tr>
<tr>
<td>Larry Lintos</td>
<td>NULL</td>
</tr>
<tr>
<td>Tim Lentleson</td>
<td>NULL</td>
</tr>
<tr>
<td>Carl Vanamos</td>
<td>NULL</td>
</tr>
<tr>
<td>Steve Mossely</td>
<td>NULL</td>
</tr>
<tr>
<td>Manny Stolanaro</td>
<td>NULL</td>
</tr>
<tr>
<td>Michael Hillson</td>
<td>NULL</td>
</tr>
<tr>
<td>Neil Gaston</td>
<td>NULL</td>
</tr>
<tr>
<td>Mo Grandosi</td>
<td>NULL</td>
</tr>
<tr>
<td>Mark Hasty</td>
<td>NULL</td>
</tr>
</tbody>
</table>

**NULLIF**
### 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:

```sql
NVL( expression_1, expression_2,...expression_n);
```

is equivalent to:

```sql
CASE WHEN expression_1 IS NOT NULL THEN expression_1
    ELSE WHEN expression_2 IS NOT NULL THEN expression_2
    ...
    ELSE expression_n;
```

**Syntax**

```sql
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.
```
-- 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;
```

<table>
<thead>
<tr>
<th>ID</th>
<th>FLDGA</th>
<th>PASSE</th>
<th>WILDP</th>
<th>PICKO</th>
<th>First</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>142</td>
<td>4</td>
<td>20</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>131</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>3</td>
<td>99</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>4</td>
<td>140</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>5</td>
<td>142</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>6</td>
<td>88</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>7</td>
<td>124</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>8</td>
<td>51</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>9</td>
<td>93</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>10</td>
<td>79</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>39</td>
<td>73</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>52</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>41</td>
<td>70</td>
<td>NULL</td>
<td>NULL</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>42</td>
<td>55</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>43</td>
<td>77</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>46</td>
<td>67</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>134</td>
<td>4</td>
<td>34</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>119</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>51</td>
<td>147</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>52</td>
<td>148</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>53</td>
<td>152</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>54</td>
<td>64</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>55</td>
<td>93</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>56</td>
<td>147</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>57</td>
<td>85</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>58</td>
<td>62</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>59</td>
<td>64</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>62</td>
<td>53</td>
<td>1</td>
<td>11</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>64</td>
<td>59</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>81</td>
<td>76</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>82</td>
<td>71</td>
<td>NULL</td>
<td>NULL</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>84</td>
<td>68</td>
<td>NULL</td>
<td>NULL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>92</td>
<td>81</td>
<td>NULL</td>
<td>NULL</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

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 (  )
```

**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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>BIRTHDATE</th>
<th>Quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Norman Aikman</td>
<td>1982-01-05</td>
<td>1</td>
</tr>
<tr>
<td>Bob Cranker</td>
<td>1987-01-21</td>
<td>1</td>
</tr>
<tr>
<td>Buddy Painter</td>
<td>1987-03-27</td>
<td>1</td>
</tr>
<tr>
<td>Jeremy Packman</td>
<td>1989-01-01</td>
<td>1</td>
</tr>
<tr>
<td>Andy Sussman</td>
<td>1990-03-22</td>
<td>1</td>
</tr>
<tr>
<td>Mitch Duffer</td>
<td>1991-01-15</td>
<td>1</td>
</tr>
<tr>
<td>Harry Pennello</td>
<td>1983-04-13</td>
<td>2</td>
</tr>
<tr>
<td>Alex Darba</td>
<td>1984-04-11</td>
<td>2</td>
</tr>
<tr>
<td>Billy Bopper</td>
<td>1988-04-20</td>
<td>2</td>
</tr>
<tr>
<td>Kelly Tamlin</td>
<td>1990-06-16</td>
<td>2</td>
</tr>
<tr>
<td>Mark Briste</td>
<td>1977-08-30</td>
<td>3</td>
</tr>
<tr>
<td>Alex Paramour</td>
<td>1981-07-02</td>
<td>3</td>
</tr>
<tr>
<td>Joseph Arkman</td>
<td>1984-09-21</td>
<td>3</td>
</tr>
<tr>
<td>Elliot Andrews</td>
<td>1989-08-21</td>
<td>3</td>
</tr>
<tr>
<td>Craig McGawn</td>
<td>1982-10-12</td>
<td>4</td>
</tr>
<tr>
<td>Jason Minman</td>
<td>1983-11-06</td>
<td>4</td>
</tr>
<tr>
<td>Greg Brown</td>
<td>1983-12-24</td>
<td>4</td>
</tr>
<tr>
<td>Henry Socomy</td>
<td>1989-11-17</td>
<td>4</td>
</tr>
<tr>
<td>John Purser</td>
<td>1990-10-30</td>
<td>4</td>
</tr>
</tbody>
</table>

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:
### Example

The following query ranks the salaries of players, per team, whose salary is at least $1 million.

<table>
<thead>
<tr>
<th>Value</th>
<th>RANK</th>
<th>DENSE_RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>e</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>
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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>TEAM</th>
<th>SEASON</th>
<th>SALARY</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mitch Hassleman</td>
<td>Cards</td>
<td>2015</td>
<td>17000000</td>
<td>1</td>
</tr>
<tr>
<td>Yuri Milleton</td>
<td>Cards</td>
<td>2015</td>
<td>15200000</td>
<td>2</td>
</tr>
<tr>
<td>James Grassler</td>
<td>Cards</td>
<td>2015</td>
<td>9375000</td>
<td>3</td>
</tr>
<tr>
<td>Jack Hellman</td>
<td>Cards</td>
<td>2015</td>
<td>8300000</td>
<td>4</td>
</tr>
<tr>
<td>Larry Lintos</td>
<td>Cards</td>
<td>2015</td>
<td>7000000</td>
<td>5</td>
</tr>
<tr>
<td>Jeremy Johnson</td>
<td>Cards</td>
<td>2015</td>
<td>4125000</td>
<td>6</td>
</tr>
<tr>
<td>Mitch Canepa</td>
<td>Cards</td>
<td>2015</td>
<td>3750000</td>
<td>7</td>
</tr>
<tr>
<td>Mitch Brandon</td>
<td>Cards</td>
<td>2015</td>
<td>3500000</td>
<td>8</td>
</tr>
<tr>
<td>Robert Cohen</td>
<td>Cards</td>
<td>2015</td>
<td>3000000</td>
<td>9</td>
</tr>
<tr>
<td>James Woegren</td>
<td>Cards</td>
<td>2015</td>
<td>2675000</td>
<td>10</td>
</tr>
<tr>
<td>Sam Culligan</td>
<td>Cards</td>
<td>2015</td>
<td>2652732</td>
<td>11</td>
</tr>
<tr>
<td>Barry Morse</td>
<td>Cards</td>
<td>2015</td>
<td>2379781</td>
<td>12</td>
</tr>
<tr>
<td>Michael Rastono</td>
<td>Cards</td>
<td>2015</td>
<td>2000000</td>
<td>13</td>
</tr>
<tr>
<td>Carl Vanamos</td>
<td>Cards</td>
<td>2015</td>
<td>2000000</td>
<td>13</td>
</tr>
<tr>
<td>Alex Wister</td>
<td>Cards</td>
<td>2015</td>
<td>1950000</td>
<td>15</td>
</tr>
<tr>
<td>Pablo Bonjorno</td>
<td>Cards</td>
<td>2015</td>
<td>1650000</td>
<td>16</td>
</tr>
<tr>
<td>Jonathan Pearlman</td>
<td>Cards</td>
<td>2015</td>
<td>1500000</td>
<td>17</td>
</tr>
<tr>
<td>Jan Bromley</td>
<td>Cards</td>
<td>2015</td>
<td>1200000</td>
<td>18</td>
</tr>
<tr>
<td>Martin Cassman</td>
<td>Giants</td>
<td>2015</td>
<td>20833333</td>
<td>1</td>
</tr>
<tr>
<td>Harry Pennello</td>
<td>Giants</td>
<td>2015</td>
<td>18500000</td>
<td>2</td>
</tr>
<tr>
<td>Tam Lassiter</td>
<td>Giants</td>
<td>2015</td>
<td>18000000</td>
<td>3</td>
</tr>
<tr>
<td>Buddy Painter</td>
<td>Giants</td>
<td>2015</td>
<td>17277777</td>
<td>4</td>
</tr>
<tr>
<td>Thomas Hillman</td>
<td>Giants</td>
<td>2015</td>
<td>12000000</td>
<td>5</td>
</tr>
<tr>
<td>Alex Paramour</td>
<td>Giants</td>
<td>2015</td>
<td>10250000</td>
<td>6</td>
</tr>
<tr>
<td>Jack Peepers</td>
<td>Giants</td>
<td>2015</td>
<td>9000000</td>
<td>7</td>
</tr>
<tr>
<td>Mark Briste</td>
<td>Giants</td>
<td>2015</td>
<td>8000000</td>
<td>8</td>
</tr>
<tr>
<td>Marcus Bamburger</td>
<td>Giants</td>
<td>2015</td>
<td>6950000</td>
<td>9</td>
</tr>
<tr>
<td>Jalen Ardson</td>
<td>Giants</td>
<td>2015</td>
<td>6000000</td>
<td>10</td>
</tr>
<tr>
<td>Steve Raster</td>
<td>Giants</td>
<td>2015</td>
<td>6000000</td>
<td>10</td>
</tr>
<tr>
<td>Sam Castleman</td>
<td>Giants</td>
<td>2015</td>
<td>5000000</td>
<td>12</td>
</tr>
<tr>
<td>Craig McGawn</td>
<td>Giants</td>
<td>2015</td>
<td>4800000</td>
<td>13</td>
</tr>
<tr>
<td>Norman Aikman</td>
<td>Giants</td>
<td>2015</td>
<td>4000000</td>
<td>14</td>
</tr>
<tr>
<td>Randy Varner</td>
<td>Giants</td>
<td>2015</td>
<td>4000000</td>
<td>14</td>
</tr>
<tr>
<td>Jason Lilliput</td>
<td>Giants</td>
<td>2015</td>
<td>4000000</td>
<td>14</td>
</tr>
<tr>
<td>Billy Bopper</td>
<td>Giants</td>
<td>2015</td>
<td>3600000</td>
<td>17</td>
</tr>
<tr>
<td>Greg Brown</td>
<td>Giants</td>
<td>2015</td>
<td>3600000</td>
<td>17</td>
</tr>
<tr>
<td>Mitch Lovell</td>
<td>Giants</td>
<td>2015</td>
<td>3578825</td>
<td>19</td>
</tr>
<tr>
<td>Bob Cranker</td>
<td>Giants</td>
<td>2015</td>
<td>3175000</td>
<td>20</td>
</tr>
<tr>
<td>Yuri Piaman</td>
<td>Giants</td>
<td>2015</td>
<td>2100000</td>
<td>21</td>
</tr>
<tr>
<td>Joseph Arkman</td>
<td>Giants</td>
<td>2015</td>
<td>1450000</td>
<td>22</td>
</tr>
<tr>
<td>Trevor Imhof</td>
<td>Giants</td>
<td>2015</td>
<td>1100000</td>
<td>23</td>
</tr>
<tr>
<td>Jason Minman</td>
<td>Giants</td>
<td>2015</td>
<td>1000000</td>
<td>24</td>
</tr>
</tbody>
</table>

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;
```

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>TEAM</th>
<th>SEASON</th>
<th>SALARY</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mitch Hassleman</td>
<td>Cards</td>
<td>2015</td>
<td>17000000</td>
<td>1</td>
</tr>
<tr>
<td>Yuri Milleton</td>
<td>Cards</td>
<td>2015</td>
<td>15200000</td>
<td>2</td>
</tr>
<tr>
<td>James Grassser</td>
<td>Cards</td>
<td>2015</td>
<td>9375000</td>
<td>3</td>
</tr>
<tr>
<td>Jack Hellman</td>
<td>Cards</td>
<td>2015</td>
<td>8300000</td>
<td>4</td>
</tr>
<tr>
<td>Larry Lintos</td>
<td>Cards</td>
<td>2015</td>
<td>7000000</td>
<td>5</td>
</tr>
<tr>
<td>Jeremy Johnson</td>
<td>Cards</td>
<td>2015</td>
<td>4125000</td>
<td>6</td>
</tr>
<tr>
<td>Mitch Canepa</td>
<td>Cards</td>
<td>2015</td>
<td>3750000</td>
<td>7</td>
</tr>
<tr>
<td>Mitch Brandon</td>
<td>Cards</td>
<td>2015</td>
<td>3500000</td>
<td>8</td>
</tr>
<tr>
<td>Robert Cohen</td>
<td>Cards</td>
<td>2015</td>
<td>3000000</td>
<td>9</td>
</tr>
<tr>
<td>James Woegren</td>
<td>Cards</td>
<td>2015</td>
<td>2675000</td>
<td>10</td>
</tr>
<tr>
<td>Sam Culligan</td>
<td>Cards</td>
<td>2015</td>
<td>2652732</td>
<td>11</td>
</tr>
<tr>
<td>Barry Morse</td>
<td>Cards</td>
<td>2015</td>
<td>2379781</td>
<td>12</td>
</tr>
<tr>
<td>Michael RaStono</td>
<td>Cards</td>
<td>2015</td>
<td>2000000</td>
<td>13</td>
</tr>
<tr>
<td>Carl Vanamos</td>
<td>Cards</td>
<td>2015</td>
<td>2000000</td>
<td>13</td>
</tr>
<tr>
<td>Alex Wister</td>
<td>Cards</td>
<td>2015</td>
<td>1950000</td>
<td>14</td>
</tr>
<tr>
<td>Pablo Bonjourno</td>
<td>Cards</td>
<td>2015</td>
<td>1650000</td>
<td>15</td>
</tr>
<tr>
<td>Jonathan Pearlman</td>
<td>Cards</td>
<td>2015</td>
<td>1500000</td>
<td>16</td>
</tr>
<tr>
<td>Jan Bromley</td>
<td>Cards</td>
<td>2015</td>
<td>1200000</td>
<td>17</td>
</tr>
<tr>
<td>Martin Cassman</td>
<td>Giants</td>
<td>2015</td>
<td>20833333</td>
<td>1</td>
</tr>
<tr>
<td>Harry Pennello</td>
<td>Giants</td>
<td>2015</td>
<td>18500000</td>
<td>2</td>
</tr>
<tr>
<td>Tam Lassiter</td>
<td>Giants</td>
<td>2015</td>
<td>18000000</td>
<td>3</td>
</tr>
<tr>
<td>Buddy Painter</td>
<td>Giants</td>
<td>2015</td>
<td>17277777</td>
<td>4</td>
</tr>
<tr>
<td>Thomas Hillman</td>
<td>Giants</td>
<td>2015</td>
<td>12000000</td>
<td>5</td>
</tr>
<tr>
<td>Alex Paramour</td>
<td>Giants</td>
<td>2015</td>
<td>10250000</td>
<td>6</td>
</tr>
<tr>
<td>Jack Peepers</td>
<td>Giants</td>
<td>2015</td>
<td>9000000</td>
<td>7</td>
</tr>
<tr>
<td>Mark Briste</td>
<td>Giants</td>
<td>2015</td>
<td>8000000</td>
<td>8</td>
</tr>
<tr>
<td>Marcus Bamburger</td>
<td>Giants</td>
<td>2015</td>
<td>6950000</td>
<td>9</td>
</tr>
<tr>
<td>Jalen Ardson</td>
<td>Giants</td>
<td>2015</td>
<td>6000000</td>
<td>10</td>
</tr>
<tr>
<td>Steve Raster</td>
<td>Giants</td>
<td>2015</td>
<td>6000000</td>
<td>10</td>
</tr>
<tr>
<td>Sam Castleman</td>
<td>Giants</td>
<td>2015</td>
<td>5000000</td>
<td>11</td>
</tr>
<tr>
<td>Craig McGawn</td>
<td>Giants</td>
<td>2015</td>
<td>4800000</td>
<td>12</td>
</tr>
<tr>
<td>Norman Aikman</td>
<td>Giants</td>
<td>2015</td>
<td>4000000</td>
<td>13</td>
</tr>
<tr>
<td>Randy Varner</td>
<td>Giants</td>
<td>2015</td>
<td>4000000</td>
<td>13</td>
</tr>
<tr>
<td>Jason Lilliput</td>
<td>Giants</td>
<td>2015</td>
<td>4000000</td>
<td>13</td>
</tr>
<tr>
<td>Billy Bopper</td>
<td>Giants</td>
<td>2015</td>
<td>3600000</td>
<td>14</td>
</tr>
<tr>
<td>Greg Brown</td>
<td>Giants</td>
<td>2015</td>
<td>3600000</td>
<td>14</td>
</tr>
<tr>
<td>Mitch Lovell</td>
<td>Giants</td>
<td>2015</td>
<td>3578825</td>
<td>15</td>
</tr>
<tr>
<td>Bob Cranker</td>
<td>Giants</td>
<td>2015</td>
<td>3175000</td>
<td>16</td>
</tr>
<tr>
<td>Yuri Piamam</td>
<td>Giants</td>
<td>2015</td>
<td>2100000</td>
<td>17</td>
</tr>
<tr>
<td>Joseph Arkman</td>
<td>Giants</td>
<td>2015</td>
<td>1450000</td>
<td>18</td>
</tr>
<tr>
<td>Trevor Imhof</td>
<td>Giants</td>
<td>2015</td>
<td>1100000</td>
<td>19</td>
</tr>
<tr>
<td>Jason Minman</td>
<td>Giants</td>
<td>2015</td>
<td>1000000</td>
<td>20</td>
</tr>
</tbody>
</table>

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**

```sql
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](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
**REPEAT**

The `REPEAT` function returns a string created by concatenating a character string value a specified number of times.

**Syntax**

```
REPEAT ( stringToRepeat, numOfRepeats )
```

`stringToRepeat`

- An expression that specifies the string to be repeated. The expression must represent a value that is of type `CHAR`, `VARCHAR`, or `LONG VARCHAR`.
- If this value is `null`, the `REPEAT` function returns `null`.

`numOfRepeats`

- A non-negative integer value that specifies the number of times to concatenate `stringToRepeat` to form the resulting value.
- If this value is `0` and `stringToRepeat` is non-null, then the `REPEAT` function returns the empty string.

**Results**

The data type of the result is the same as the type of the `stringToRepeat` argument. The width of the result is calculated as follows:

- If `numOfRepeats` is a known constant at bind time, the width of the result is:
  \[(\text{maxWidth of } \text{stringToRepeat}) \times \text{numOfRepeats}\]

- If `numOfRepeats` is an expression whose value is either dynamic or unknown at bind time (e.g. a parameter), the width of the result is:
  The maximum possible width of the type of the `stringToRepeat` argument.

**Examples**

We create a simple table for the examples in this section:

```
splice> CREATE TABLE t1 (a1 CHAR(5), b1 VARCHAR(5), c1 INT);
0 rows inserted/updated/deleted

splice> INSERT INTO t1 VALUES ('aaa', 'aaa', 3), ('bbbb', 'bbbb', 4);
2 rows inserted/updated/deleted
```
**Example 1: Repeat Count is a Constant**

```sql
splice> select repeat(a1, 3) from t1;
----------------------------------------------------
aaa  aaa  aaa
bbbb bbbb bbbb
2 rows selected
```

**Example 2: Repeat Count is Parameterized**

```sql
prepare q1 as 'select repeat(a1, ?) from t1';
splice> execute q1 using 'values (3)';
1
----------------------------------------------------
aaa  aaa  aaa
bbbb bbbb bbbb
2 rows selected
```

**Example 3: Both Arguments are Expressions**

```sql
splice> select repeat('-', || a1 || b1 || '-', c1) from t1;
1
----------------------------------------------------
-aaa  aaa--aaa  aaa--aaa  aaa-  
-bbbb bbbb--bbbb bbbb--bbbb bbbb--bbbb bbbb--
2 rows selected
```
**REPLACE**

The **REPLACE** function replaces all occurrences of a substring within a string and returns the new string.

**Syntax**

The syntax for the **REPLACE** function is:

```
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 example 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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>AVERAGE</th>
<th>TEAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buddy Painter</td>
<td>0.31777</td>
<td>Giants</td>
</tr>
<tr>
<td>John Purser</td>
<td>0.31151</td>
<td>Giants</td>
</tr>
<tr>
<td>Kelly Tamlin</td>
<td>0.30337</td>
<td>Giants</td>
</tr>
<tr>
<td>Stan Post</td>
<td>0.30472</td>
<td>Cards</td>
</tr>
</tbody>
</table>

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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>AVERAGE</th>
<th>TRADED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stan Post</td>
<td>0.30472</td>
<td>Giants</td>
</tr>
</tbody>
</table>

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
**RIGHT**

The **RIGHT** function returns the rightmost n characters from a string.

**Syntax**

```sql
RIGHT( string, n )
```

- **string**
  - The string from which you want to extract characters.
- **n**
  - An integer value that specifies the number of characters that you want to extract from **string**.

**Results**

This function takes as arguments a string from which you want to extract a substring, working backwards from the last (rightmost) character in the string, and an integer that specifies the number of characters that you want to extract.

If there are fewer characters in the **string** than are specified in **n**, the entire **string** is returned (no padding is applied to the result).

**Examples**

```sql
splice> VALUES( RIGHT( 'Splice Machine', 6 ) );
1
-----------------
achine

splice> VALUES( RIGHT( 'Splice Machine', 3 ) );
1
-----------------
ine

splice> VALUES( RIGHT( 'Splice Machine', 32 ) );
1
-----------------
Splice Machine
```
ROUND

The ROUND function rounds the specified number up or down to the (optionally) specified number of decimal places.

Syntax

```
ROUND ( num, decimals )
```

`num`
A DOUBLE PRECISION number that is the numeric value you want rounded.

`decimals`
Optional. An integer value that specifies the number of decimal digits to which you want `number` rounded.

If you do not specify `decimals`, then `num` is rounded to an integer value.

If `decimals` is a negative number, then `num` is rounded to that number of digits to the left of the decimal point.
Examples

splice>VALUES ROUND(84.4);
1
--------
84

splice>VALUES ROUND(84.4, 1);
1
--------
84.4

1 row selected

splice>VALUES ROUND(84.4, 0);
1
--------
84.0

1 row selected

splice>VALUES ROUND(84.4, -1);
1
--------
80.0

1 row selected

splice>VALUES ROUND(844.4, -1);
1
--------
840.0

1 row selected

splice>VALUES ROUND(844.4, -2);
1
--------
800.0

1 row selected
Results
The data type of the result is a DOUBLE PRECISION number.

- 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
**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:

```sql
splice> SELECT ROWID, DisplayName, Position
    FROM Players
    WHERE Team='Giants' and Position='OF';
```

<table>
<thead>
<tr>
<th>ROWID</th>
<th>DISPLAYNAME</th>
<th>POS&amp;</th>
</tr>
</thead>
<tbody>
<tr>
<td>89</td>
<td>Greg Brown</td>
<td>OF</td>
</tr>
<tr>
<td>93</td>
<td>Jeremy Packman</td>
<td>OF</td>
</tr>
<tr>
<td>95</td>
<td>Jason Pratter</td>
<td>OF</td>
</tr>
<tr>
<td>99</td>
<td>Reed Lister</td>
<td>OF</td>
</tr>
</tbody>
</table>

4 rows selected

This statement updates column `c` in all rows in which column `b` equals 10:

```sql
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**

```sql
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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>SALARY</th>
<th>RowNum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mitch Hassleman</td>
<td>17000000</td>
<td>1</td>
</tr>
<tr>
<td>Yuri Milleton</td>
<td>15200000</td>
<td>2</td>
</tr>
<tr>
<td>James Grasser</td>
<td>9375000</td>
<td>3</td>
</tr>
<tr>
<td>Jack Hellman</td>
<td>8300000</td>
<td>4</td>
</tr>
<tr>
<td>Larry Lintos</td>
<td>7000000</td>
<td>5</td>
</tr>
<tr>
<td>Jeremy Johnson</td>
<td>4125000</td>
<td>6</td>
</tr>
<tr>
<td>Mitch Canepa</td>
<td>3750000</td>
<td>7</td>
</tr>
<tr>
<td>Mitch Brandon</td>
<td>3500000</td>
<td>8</td>
</tr>
<tr>
<td>Robert Cohen</td>
<td>3000000</td>
<td>9</td>
</tr>
<tr>
<td>James Woegren</td>
<td>2675000</td>
<td>10</td>
</tr>
<tr>
<td>Sam Culligan</td>
<td>2652732</td>
<td>11</td>
</tr>
<tr>
<td>Barry Morse</td>
<td>2379781</td>
<td>12</td>
</tr>
<tr>
<td>Michael Rastono</td>
<td>2000000</td>
<td>13</td>
</tr>
<tr>
<td>Carl Vanamos</td>
<td>2000000</td>
<td>14</td>
</tr>
<tr>
<td>Alex Wister</td>
<td>1950000</td>
<td>15</td>
</tr>
<tr>
<td>Pablo Bonjourno</td>
<td>1650000</td>
<td>16</td>
</tr>
<tr>
<td>Jonathan Pearlman</td>
<td>1500000</td>
<td>17</td>
</tr>
<tr>
<td>Jan Bromley</td>
<td>1200000</td>
<td>18</td>
</tr>
</tbody>
</table>

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 NULL value.

Example

splice> VALUES( NOW(), HOUR(NOW), MINUTE(NOW), SECOND(NOW) );

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015-11-12 17:48:55.217</td>
<td>17</td>
<td>48</td>
<td>55.217</td>
</tr>
</tbody>
</table>

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**

```sql
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**

```sql
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**

```sql
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
» CO\text{T} function
» DEGREES function
» RADIANS function
» SINH function
» TAN function
» TANH function
**SINH**

The SINH function returns the hyperbolic sine of a specified number.

**Syntax**

```sql
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**

```sql
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;

<table>
<thead>
<tr>
<th>ID</th>
<th>ERA</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>3</td>
</tr>
<tr>
<td>34</td>
<td>5</td>
</tr>
<tr>
<td>36</td>
<td>3</td>
</tr>
<tr>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
</tr>
<tr>
<td>42</td>
<td>2</td>
</tr>
<tr>
<td>44</td>
<td>5</td>
</tr>
<tr>
<td>46</td>
<td>2</td>
</tr>
<tr>
<td>48</td>
<td>5</td>
</tr>
<tr>
<td>72</td>
<td>2</td>
</tr>
<tr>
<td>74</td>
<td>3</td>
</tr>
<tr>
<td>76</td>
<td>2</td>
</tr>
<tr>
<td>78</td>
<td>3</td>
</tr>
<tr>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>82</td>
<td>3</td>
</tr>
<tr>
<td>84</td>
<td>2</td>
</tr>
<tr>
<td>86</td>
<td>2</td>
</tr>
<tr>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td>90</td>
<td>2</td>
</tr>
<tr>
<td>92</td>
<td>2</td>
</tr>
<tr>
<td>94</td>
<td>2</td>
</tr>
</tbody>
</table>

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 | 9.48632980505138
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 | 12.28820572444508
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
### 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:
```sql
splice> SELECT AVG(Salary) as AvgSalary, STDDEV_POP(Salary) AS PopStdDev, STDDEV_SAMP(Salary) AS SampStdDev FROM Salaries;

<table>
<thead>
<tr>
<th>AVG_SALARY</th>
<th>POP_STD_DEV</th>
<th>SAM_STD_DEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>2949737</td>
<td>4694155.715951055</td>
<td>4719325.63212163</td>
</tr>
</tbody>
</table>

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;

<table>
<thead>
<tr>
<th>AVGSALARY</th>
<th>POPSTDDEV</th>
<th>SAMPSTDDEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>2949737</td>
<td>4694155.715951055</td>
<td>4719325.63212163</td>
</tr>
</tbody>
</table>

1 row selected
**STRIP**

The **STRIP** function 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
- which pad character to remove

**Syntax**

```
STRIP( stripSource { ',' striptype } { ',', stripCharacter } )
```

*stripSource*
- The character expression to be stripped

*stripType*
- Whether to strip from the left end, the right end, or both ends of the *stripSource* string. Specify this value unquoted, as shown in the examples below. You can use the full or short identifiers:

| LEADING or L | Strip characters from the left side of the string. |
| TRAILING or T | Strip characters from the right side of the string. |
| BOTH or B     | Strip characters from both the left and right sides of the string. |

If you don’t specify a *stripType* value, the default value of **BOTH** is used.

*stripCharacter*
- A character expression, enclosed in single quotes, that specifies which character to strip 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 (` ' `).

**Results**

If either *stripCharacter* or *stripSource* evaluates to **NULL**, the result of the **STRIP** function is **NULL**. Otherwise, the result is defined as follows:

- If *stripType* is **LEADING**, the result will be the *stripSource* value with all leading occurrences of *stripCharacter* removed.
- If *stripType* is **TRAILING**, the result will be the *stripSource* value with all trailing occurrences of *stripCharacter* removed.
If `stripType` is **BOTH**, the result will be the `stripSource` value with all leading AND trailing occurrences of `stripCharacter` removed.

If `stripSource`'s data type is **CHAR** or **VARCHAR**, the return type of the `STRIP` function will be **VARCHAR**. Otherwise the return type of the `STRIP` function will be **CLOB**.
Examples

splice> values strip(' space case ', b);
1
----------------
space case

splice> values strip(' space case ', both);
1
----------------
space case

splice> values strip(' space case ', L);
1
----------------
space case

splice> values strip(' space case ', LEADING);
1
----------------
space case

splice> values strip(' space case ', t);
1
----------------
space case

splice> values strip(' space case ', TRAILING);
1
----------------
space case

splice> values strip('aabbccaa', b, 'a');
1
--------
bbcc

splice> values strip('aabbccaa', l, 'a');
1
--------
bbccaa

splice> values strip('aabbccaa', t, 'a');
1
--------
aabbcc
See Also

» About Data Types
» Concatenation operator
» INITCAP function
» INSTR function
» LCASE function
» LENGTH function
» LOCATE function
» REGEX_LIKE operator
» REPLACE function
» SUBSTR function
» UCASE function
**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 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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>1To4</th>
<th>5ToEnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buddy Painter</td>
<td>Buddy</td>
<td>dy Painter</td>
</tr>
<tr>
<td>Billy Bopper</td>
<td>Bill</td>
<td>ly Bopper</td>
</tr>
<tr>
<td>John Purser</td>
<td>John</td>
<td>n Purser</td>
</tr>
<tr>
<td>Bob Cranker</td>
<td>Bob</td>
<td>Cranker</td>
</tr>
<tr>
<td>Mitch Duffer</td>
<td>Mitch</td>
<td>ch Duffer</td>
</tr>
<tr>
<td>Norman Aikman</td>
<td>Norm</td>
<td>man Aikman</td>
</tr>
<tr>
<td>Alex Paramour</td>
<td>Alex</td>
<td>x Paramour</td>
</tr>
<tr>
<td>Harry Pennello</td>
<td>Harry</td>
<td>ry Pennello</td>
</tr>
<tr>
<td>Greg Brown</td>
<td>Greg</td>
<td>g Brown</td>
</tr>
<tr>
<td>Jason Minman</td>
<td>Jason</td>
<td>on Minman</td>
</tr>
</tbody>
</table>

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` 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.

```sql
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;

<table>
<thead>
<tr>
<th>TEAM</th>
<th>DISPLAYNAME</th>
<th>SALARY</th>
<th>RUNNING TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cards</td>
<td>Larry Lintos</td>
<td>7000000</td>
<td>7000000</td>
</tr>
<tr>
<td>Cards</td>
<td>Jack Hellman</td>
<td>8300000</td>
<td>15300000</td>
</tr>
<tr>
<td>Cards</td>
<td>James Grasser</td>
<td>9375000</td>
<td>24675000</td>
</tr>
<tr>
<td>Cards</td>
<td>Yuri Milleton</td>
<td>15200000</td>
<td>39875000</td>
</tr>
<tr>
<td>Cards</td>
<td>Mitch Hassleman</td>
<td>17000000</td>
<td>56875000</td>
</tr>
<tr>
<td>Giants</td>
<td>Jalen Ardson</td>
<td>6000000</td>
<td>6000000</td>
</tr>
<tr>
<td>Giants</td>
<td>Steve Raster</td>
<td>6000000</td>
<td>12000000</td>
</tr>
<tr>
<td>Giants</td>
<td>Marcus Bamburger</td>
<td>6950000</td>
<td>18950000</td>
</tr>
<tr>
<td>Giants</td>
<td>Mark Briste</td>
<td>8000000</td>
<td>26950000</td>
</tr>
<tr>
<td>Giants</td>
<td>Jack Peepers</td>
<td>9000000</td>
<td>35950000</td>
</tr>
<tr>
<td>Giants</td>
<td>Alex Paramour</td>
<td>10250000</td>
<td>46200000</td>
</tr>
<tr>
<td>Giants</td>
<td>Thomas Hillman</td>
<td>12000000</td>
<td>58200000</td>
</tr>
<tr>
<td>Giants</td>
<td>Buddy Painter</td>
<td>17277777</td>
<td>75477777</td>
</tr>
<tr>
<td>Giants</td>
<td>Tam Lassiter</td>
<td>18000000</td>
<td>93477777</td>
</tr>
<tr>
<td>Giants</td>
<td>Harry Pennello</td>
<td>18500000</td>
<td>111977777</td>
</tr>
<tr>
<td>Giants</td>
<td>Martin Cassman</td>
<td>20833333</td>
<td>132811110</td>
</tr>
</tbody>
</table>

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**

\[
\text{TANH ( number )}
\]

*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 )
```

Splice Machine displays `TIME` and `TIMESTAMP` values using the current time zone for the server to which you are connected. `DATE` values, however, are stored as fixed strings, and are not affected by the time zone of the server to which you are connected.

**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 `NULL` value.
- 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

- CURRENT_DATE
- CURRENT_TIME
- DATE type
- DATE function
- DAY
- EXTRACT
- LASTDAY
- MONTH
- MONTH_BETWEEN
- MONTHNAME
- NEXTDAY
- NOW
- QUARTER
- TIME type
- TIME function
- TIMESTAMP type
- TIMESTAMP function
- TO_CHAR
- TO_DATE
- WEEK
- Working with Dates
**TIMESTAMP**

The `TIMESTAMP` function returns a timestamp from a value or a pair of values.

**Syntax**

```sql
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
- CURRENT_TIME
- DATE type
- DATE function
- DAY
- EXTRACT
- LASTDAY
MONTH
MONTH_BETWEEN
MONTHNAME
NEXTDAY
NOW
QUARTER
TIME type
TIME function
TIMESTAMP type
TO_CHAR
TO_DATE
WEEK
Working with Dates
**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**

```sql
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
-------------------------------
1 row selected
```

See Also

- `CURRENT_DATE`
- `CURRENT_TIME`
- `DATE` type
- `DATE` function
- `DAY`
- `EXTRACT`
- `LASTDAY`
- `MONTH`
- `MONTH_BETWEEN`
- `MONTHNAME`
- `NEXTDAY`
- `NOW`
- `QUARTER`
- `TIME` type
- `TIME` function
- `TIMESTAMP` type
- `TIMESTAMP` function
- `TIMESTAMPDIFF` function
- `TO_CHAR`
- `TO_DATE`
WEEK

Working with Dates
TIMESTAMPDIFF

The TIMESTAMPDIFF function finds the difference between two timestamps, in terms of the specified interval.

Syntax

```sql
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;

<table>
<thead>
<tr>
<th>ID</th>
<th>BIRTHDATE</th>
<th>YearsSinceJFK</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1981-07-02</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>1982-01-05</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>1983-04-13</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>1983-11-06</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>1983-12-24</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>1987-01-21</td>
<td>23</td>
</tr>
<tr>
<td>1</td>
<td>1987-03-27</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>1988-04-20</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>1990-10-30</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>1991-01-15</td>
<td>27</td>
</tr>
</tbody>
</table>

10 rows selected

See Also
- CURRENT_DATE
- CURRENT_TIME
- DATE type
- DATE function
- DAY
- EXTRACT
- LASTDAY
- MONTH
- MONTH_BETWEEN
- MONTHNAME
- NEXTDAY
- NOW
- QUARTER
- TIME type
» **TIME** function

» **TIMESTAMP** type

» **TIMESTAMP** function

» **TIMESTAMPADD** function

» **TO_CHAR**

» **TO_DATE**

» **WEEK**

» **Working with Dates**
TINYINT

The TINYINT function returns a tiny integer (1 byte) representation of a number or character string, in the form of a small integer constant.

Syntax

TINYINT ( 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 tiny integers. The character string cannot be a long string.

Results

The result of the function is a TINYINT. 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 tiny 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 tiny integer column or variable.

Examples

Using the Pitching table from our Doc Examples database, select the Era column in tiny integer form for further processing in the application:
splice> SELECT ID, TINYINT(Era) "ERA"  
    FROM Pitching  
    WHERE MOD(ID,2) = 0;

<table>
<thead>
<tr>
<th>ID</th>
<th>ERA</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>3</td>
</tr>
<tr>
<td>34</td>
<td>5</td>
</tr>
<tr>
<td>36</td>
<td>3</td>
</tr>
<tr>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
</tr>
<tr>
<td>42</td>
<td>2</td>
</tr>
<tr>
<td>44</td>
<td>5</td>
</tr>
<tr>
<td>46</td>
<td>2</td>
</tr>
<tr>
<td>48</td>
<td>5</td>
</tr>
<tr>
<td>72</td>
<td>2</td>
</tr>
<tr>
<td>74</td>
<td>3</td>
</tr>
<tr>
<td>76</td>
<td>2</td>
</tr>
<tr>
<td>78</td>
<td>3</td>
</tr>
<tr>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>82</td>
<td>3</td>
</tr>
<tr>
<td>84</td>
<td>2</td>
</tr>
<tr>
<td>86</td>
<td>2</td>
</tr>
<tr>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td>90</td>
<td>2</td>
</tr>
<tr>
<td>92</td>
<td>2</td>
</tr>
<tr>
<td>94</td>
<td>2</td>
</tr>
</tbody>
</table>

23 rows selected

**See Also**

- About Data Types
**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 parses a datetime string according to a formatting specification, and returns a `DATE` value. Note that the input string can represent a timestamp or date; however, the input string *must* match the formatting specification string.

**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 `DateTimeFormatter` class.

Date and time value formats are used for both parsing input values and pattern types. For example, the format specification `yyyy-MM-dd HH:mm:ssZ` parses 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.
- [Pattern 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 pattern types.
<table>
<thead>
<tr>
<th>Pattern Letter</th>
<th>Meaning</th>
<th>Pattern Type</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Era</td>
<td>Text</td>
<td>AD; Anno Domini; A</td>
</tr>
<tr>
<td>u</td>
<td>Year</td>
<td>Year</td>
<td>2017; 17</td>
</tr>
<tr>
<td>y</td>
<td>Year-of-era</td>
<td>Year</td>
<td>2017; 17</td>
</tr>
<tr>
<td>D</td>
<td>Day-of-year</td>
<td>Number</td>
<td>189; 303</td>
</tr>
<tr>
<td>M/L</td>
<td>Month-of-year</td>
<td>Number/text</td>
<td>6; 06; Jun; June</td>
</tr>
<tr>
<td>d</td>
<td>Day-of-month</td>
<td>Number</td>
<td>12</td>
</tr>
<tr>
<td>Y</td>
<td>Week-based-year</td>
<td>year</td>
<td>2017; 17</td>
</tr>
<tr>
<td>w</td>
<td>Week-of-week-based-year</td>
<td>Number</td>
<td>14; 51</td>
</tr>
<tr>
<td>W</td>
<td>Week-of-month</td>
<td>Number</td>
<td>7</td>
</tr>
<tr>
<td>E</td>
<td>Day-of-week</td>
<td>Text</td>
<td>Mon; Monday; M</td>
</tr>
<tr>
<td>e/c</td>
<td>Localized day-of-week</td>
<td>Number/text</td>
<td>1; 01; Mon; Monday; M</td>
</tr>
<tr>
<td>F</td>
<td>Week-of-month</td>
<td>Number</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>Am-Pm-of-Day</td>
<td>Text</td>
<td>PM</td>
</tr>
<tr>
<td>h</td>
<td>Clock-hour-of-am-pm (1-12)</td>
<td>Number</td>
<td>7; 12</td>
</tr>
<tr>
<td>K</td>
<td>Hour-of-am-pm (0-11)</td>
<td>Number</td>
<td>0; 11</td>
</tr>
<tr>
<td>k</td>
<td>Clock-hour-of-am-pm (1-24)</td>
<td>Number</td>
<td>1; 13</td>
</tr>
<tr>
<td>H</td>
<td>Hour-of-day (0-23)</td>
<td>Number</td>
<td>0; 11; 17</td>
</tr>
<tr>
<td>m</td>
<td>Minute-of-hour</td>
<td>Number</td>
<td>27</td>
</tr>
<tr>
<td>s</td>
<td>Second-of-minute</td>
<td>Number</td>
<td>48</td>
</tr>
<tr>
<td>S</td>
<td>Fraction-of-second</td>
<td>Fraction</td>
<td>978</td>
</tr>
<tr>
<td>A</td>
<td>Milli-of-day</td>
<td>Number</td>
<td>1234</td>
</tr>
<tr>
<td>V</td>
<td>Time-zone ID</td>
<td>Zone-id</td>
<td>America/San_Francisco; Z; -08:30</td>
</tr>
<tr>
<td>Pattern Letter</td>
<td>Meaning</td>
<td>Pattern Type</td>
<td>Example(s)</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------</td>
<td>--------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>z</td>
<td>Time-zone name</td>
<td>Zone-name</td>
<td>Pacific Standard Time; PST</td>
</tr>
<tr>
<td>O</td>
<td>Localized zone-offset</td>
<td>Offset-0</td>
<td>GMT+7; GMT+07:00; UTC-07:00;</td>
</tr>
<tr>
<td>X</td>
<td>Zone-offset 'Z' for zero</td>
<td>Offset-X</td>
<td>z; -07; -0730; -07:30; -073015; -07:30:15;</td>
</tr>
<tr>
<td>x</td>
<td>Zone-offset</td>
<td>Offset-x</td>
<td>+0000; -07; -0730; -07:30; -073015; -07:30:15;</td>
</tr>
<tr>
<td>z</td>
<td>Zone-offset</td>
<td>Offset-Z</td>
<td>+0000; -0800; -08:00;</td>
</tr>
<tr>
<td>p</td>
<td>Pad next</td>
<td>Pad modifier</td>
<td>1</td>
</tr>
<tr>
<td>'</td>
<td>Escape for text</td>
<td>Delimiter</td>
<td></td>
</tr>
<tr>
<td>‘’</td>
<td>Single quote</td>
<td>Literal</td>
<td>‘’</td>
</tr>
</tbody>
</table>

**NOTE:** The format characters x, X, and Z all allow the matching of any possible timezone offset, and all cause the same behavior.

**Pattern Types**

How a pattern 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.

<table>
<thead>
<tr>
<th>Pattern Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Text</strong></td>
<td>The text style is determined based on the number of pattern letters used;</td>
</tr>
<tr>
<td></td>
<td>» Less than 4 pattern letters will use the short form.</td>
</tr>
<tr>
<td></td>
<td>» Exactly 4 pattern letters will use the full form.</td>
</tr>
<tr>
<td></td>
<td>» Exactly 5 pattern letters will use the narrow form. Pattern letters l, c, and q specify the stand-alone form of the text styles.</td>
</tr>
<tr>
<td><strong>Pattern Type</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
| **Number**      | If the count of letters is one, then the value is parsed using the minimum number of digits and without padding. Otherwise, the count of digits is used as the width of the field, with the value zero-padded as necessary.  
The following pattern letters have constraints on the count of letters: 
  - Only one letter of c and f can be specified.  
  - Up to two letters of d, D, h, H, k, K, m, and s can be specified.  
  - Up to three letters of D can be specified. |
| **Number/Text** | If the count of pattern letters is 3 or greater, use the Text rules above. Otherwise use the Number rules above. |
| **Fraction**    | Specifying any number of fractional seconds ('S') will accept between 1 and 6 fractional seconds digits. |
| **Year**        | The count of letters determines the minimum field width below which padding is used.  
  - If the count of letters is two, then a reduced two digit form is used. This will parse using the base value of 2000 to 2099 inclusive.  
  - If the count of letters is less than four (but not two), then the sign is only parsed for negative years. Otherwise, the sign is parse if the pad width is exceeded. |
| **ZoneId**      | This specifies the time-zone ID, such as Europe/Paris. If the count of letters is two, then the time-zone ID is used. Any other count of letters results in a syntax error. |
| **Zone names**  | This specifies the display name of the time-zone ID. If the count of letters is one, two or three, then the short name is used. If the count of letters is four, then the full name is used. Five or more letters results in a syntax error. |
| **Offset O**    | This formats the localized offset based on the number of pattern letters.  
  - One letter specifies the short form of the localized offset, which is localized offset text, such as GMT, with hour without leading zero, optional 2-digit minute and second if non-zero, and colon, for example GMT+8.  
  - Four letters specifies the full form, which is localized offset text, such as GMT, with 2-digit hour and minute field, optional second field if non-zero, and colon, for example GMT+08:00.  
  - Any other count of letters results in a syntax error. |
<table>
<thead>
<tr>
<th>Pattern Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset Z</td>
<td>This formats the offset based on the number of pattern letters.</td>
</tr>
<tr>
<td></td>
<td>One, two or three letters specifies the hour and minute, without a colon, such as +0130. This will be +0000 when the offset is zero.</td>
</tr>
<tr>
<td></td>
<td>Four letters specifies the full form of localized offset, equivalent to four letters of Offset-O. This will be the corresponding localized offset text if the offset is zero.</td>
</tr>
<tr>
<td></td>
<td>Five letters specifies the hour, minute, with optional second if non-zero, with colon. It specifies Z if the offset is zero.</td>
</tr>
<tr>
<td></td>
<td>Six or more letters results in a syntax error.</td>
</tr>
<tr>
<td>Optional section</td>
<td>The optional section markers work exactly like calling DateTimeFormatterBuilder.optionalStart() and DateTimeFormatterBuilder.optionalEnd().</td>
</tr>
<tr>
<td>Pad modifier</td>
<td>Modifies the pattern that immediately follows to be padded with spaces. The pad width is determined by the number of pattern letters. This is the same as calling DateTimeFormatterBuilder.padNext(int).</td>
</tr>
<tr>
<td></td>
<td>For example, ppH specifies the hour-of-day padded on the left with spaces to a width of 2.</td>
</tr>
<tr>
<td></td>
<td>Any unrecognized letter is an error.</td>
</tr>
<tr>
<td></td>
<td>Any non-letter character, other than [, ], {, }, # and the single quote is parsed. Despite this, it is recommended to use single quotes around all characters that you want to parse directly to ensure that future changes do not break your application.</td>
</tr>
</tbody>
</table>

**Formatting Examples**
The following table contains a number of examples of date time formats:

<table>
<thead>
<tr>
<th>Date and Time Pattern</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;yyyy.MM.dd G 'at' HH:mm:ss z&quot;</td>
<td>2001.07.04 AD at 12:08:56 PDT</td>
</tr>
<tr>
<td>&quot;EEE, MMM d, 'yy&quot;</td>
<td>Wed, Jul 4, '01</td>
</tr>
<tr>
<td>&quot;h:mm a&quot;</td>
<td>12:08 PM</td>
</tr>
</tbody>
</table>
### Usage Notes

You can supply a string formatted as a timestamp to `TO_DATE`; it will translate that value into a date. Note, however, that it cannot translate a time string into a date value.

You'll note that our examples use lowercase year (e.g. `yyyy`) and day (e.g. `dd`) formats; **not** uppercase (e.g. `YYYY` or `DD`) formats. It's easy to get confused, so remember:

The uppercase `YYYY` format, which is not commonly used, specifies week year, which means that you **must** also specify a week-of-week-based-year (`w`) format value.

Similarly, the uppercase `D` format, also uncommon, specifies day-of-the-year, not the day of the month.

### Examples of Using `TO_DATE`

Here are several simple examples:
splice> VALUES TO_DATE('2015-01-01', 'yyy-MM-dd');
1
----------
2015-01-01
splice> VALUES TO_DATE('01-01-2015', 'MM-dd-yyyy');
1
----------
2015-01-01
splice> VALUES TO_DATE('2013-09-18T11:03:30.000EDT','yyy-MM-dd''T''HH:mm:ss.SSSz');
1
----------
2013-09-18
splice> VALUES (TO_DATE('01-01-2015', 'MM-dd-yyyy') + 30);
1
----------
2015-01-31

The following 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.

Note that when you specify a zone offset (Z) or time zone (z), Splice Machine interprets the timestamp in its given zone, and then adjusts the time to the time zone setting of the operating system. This means that when a timestamp value specified in the Easter time zone is parsed on an operating system based in the Pacific time zone, it will be adjusted back by three hours; for example:

splice> VALUES TO_DATE('2013-06-18T01:03:30.000EDT','yyy-MM-dd''T''HH:mm:ss.SSSz');
1
----------
2013-06-17

See Also

» CURRENT_DATE
» DATE data type
» DATE
» DAY
» EXTRACT
» LASTDAY
» MONTH
MONTH_BETWEEN
MONTHNAME
NEXTDAY
NOW
QUARTER
TIME data type
TIMESTAMP
TO_CHAR
TO_TIME
TO_TIMESTAMP
WEEK

Working with Dates in the Developer's Guide
**TO_TIME**

The TO_TIME function parses a datetime string according to a formatting specification, and returns a TIME value. Note that the input string can represent a timestamp or time; however, the input string must match the formatting specification string.

**Syntax**

```
TO_TIME( timeStrExpr, formatStr );
```

*timeStrExpr*  
A string expression that contains a time that is formatted according to the format string.

*formatStr*  
A string that specifies the format you want applied to the timeStr. See the Date and Time Formats section below for more information about format specification.

**Results**

The result is always a TIME value.

**Date and Time Formats**

Splice Machine supports date and time format specifications based on the Java DateTimeFormatter class.

Date and time value formats are used for both parsing input values and pattern types. For example, the format specification `yyyy-MM-dd HH:mm:ssZ` parses 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.
- **Pattern 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 pattern types.
<table>
<thead>
<tr>
<th>Pattern Letter</th>
<th>Meaning</th>
<th>Pattern Type</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Era</td>
<td>Text</td>
<td>AD; Anno Domini; A</td>
</tr>
<tr>
<td>u</td>
<td>Year</td>
<td>Year</td>
<td>2017; 17</td>
</tr>
<tr>
<td>y</td>
<td>Year-of-era</td>
<td>Year</td>
<td>2017; 17</td>
</tr>
<tr>
<td>D</td>
<td>Day-of-year</td>
<td>Number</td>
<td>189; 303</td>
</tr>
<tr>
<td>M/L</td>
<td>Month-of-year</td>
<td>Number/text</td>
<td>6; 06; Jun; June</td>
</tr>
<tr>
<td>d</td>
<td>Day-of-month</td>
<td>Number</td>
<td>12</td>
</tr>
<tr>
<td>Y</td>
<td>Week-based-year</td>
<td>year</td>
<td>2017; 17</td>
</tr>
<tr>
<td>w</td>
<td>Week-of-week-based-year</td>
<td>Number</td>
<td>14; 51</td>
</tr>
<tr>
<td>W</td>
<td>Week-of-month</td>
<td>Number</td>
<td>7</td>
</tr>
<tr>
<td>E</td>
<td>Day-of-week</td>
<td>Text</td>
<td>Mon; Monday; M</td>
</tr>
<tr>
<td>e/c</td>
<td>Localized day-of-week</td>
<td>Number/text</td>
<td>1; 01; Mon; Monday; M</td>
</tr>
<tr>
<td>F</td>
<td>Week-of-month</td>
<td>Number</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>Am-Pm-of-Day</td>
<td>Text</td>
<td>PM</td>
</tr>
<tr>
<td>h</td>
<td>Clock-hour-of-am-pm (1-12)</td>
<td>Number</td>
<td>7; 12</td>
</tr>
<tr>
<td>K</td>
<td>Hour-of-am-pm (0-11)</td>
<td>Number</td>
<td>0; 11</td>
</tr>
<tr>
<td>k</td>
<td>Clock-hour-of-am-pm (1-24)</td>
<td>Number</td>
<td>1; 13</td>
</tr>
<tr>
<td>H</td>
<td>Hour-of-day (0-23)</td>
<td>Number</td>
<td>0; 11; 17</td>
</tr>
<tr>
<td>m</td>
<td>Minute-of-hour</td>
<td>Number</td>
<td>27</td>
</tr>
<tr>
<td>s</td>
<td>Second-of-minute</td>
<td>Number</td>
<td>48</td>
</tr>
<tr>
<td>S</td>
<td>Fraction-of-second</td>
<td>Fraction</td>
<td>978</td>
</tr>
<tr>
<td>A</td>
<td>Milli-of-day</td>
<td>Number</td>
<td>1234</td>
</tr>
<tr>
<td>V</td>
<td>Time-zone ID</td>
<td>Zone-id</td>
<td>America/San_Francisco; Z; -08:30</td>
</tr>
<tr>
<td>Pattern Letter</td>
<td>Meaning</td>
<td>Pattern Type</td>
<td>Example(s)</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------</td>
<td>--------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>z</td>
<td>Time-zone name</td>
<td>Zone-name</td>
<td>Pacific Standard Time; PST</td>
</tr>
<tr>
<td>O</td>
<td>Localized zone-offset</td>
<td>Offset-0</td>
<td>GMT+7; GMT+07:00; UTC-07:00;</td>
</tr>
<tr>
<td>X</td>
<td>Zone-offset 'Z' for zero</td>
<td>Offset-X</td>
<td>Z; -07; -0730; -07:30; -0730:15;</td>
</tr>
<tr>
<td>x</td>
<td>Zone-offset</td>
<td>Offset-x</td>
<td>+0000; -07; -0730; -07:30; -0730:15;</td>
</tr>
<tr>
<td>Z</td>
<td>Zone-offset</td>
<td>Offset-Z</td>
<td>+0000; -0800; -08:00;</td>
</tr>
<tr>
<td>p</td>
<td>Pad next</td>
<td>Pad modifier</td>
<td>1</td>
</tr>
<tr>
<td>'</td>
<td>Escape for text</td>
<td>Delimiter</td>
<td></td>
</tr>
<tr>
<td>' '</td>
<td>Single quote</td>
<td>Literal</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** The format characters x, X, and Z all allow the matching of any possible timezone offset, and all cause the same behavior.

**Pattern Types**

How a pattern 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.

<table>
<thead>
<tr>
<th>Pattern Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Text</em></td>
<td>The text style is determined based on the number of pattern letters used;</td>
</tr>
<tr>
<td></td>
<td>〉 Less than 4 pattern letters will use the short form.</td>
</tr>
<tr>
<td></td>
<td>〉 Exactly 4 pattern letters will use the full form.</td>
</tr>
<tr>
<td></td>
<td>〉 Exactly 5 pattern letters will use the narrow form. Pattern letters L, c, and q specify the stand-alone form of the text styles.</td>
</tr>
<tr>
<td>Pattern Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Number</strong></td>
<td>If the count of letters is one, then the value is parsed using the minimum number of digits and without padding. Otherwise, the count of digits is used as the width of the field, with the value zero-padded as necessary.</td>
</tr>
<tr>
<td></td>
<td>The following pattern letters have constraints on the count of letters:</td>
</tr>
<tr>
<td></td>
<td>» Only one letter of <code>c</code> and <code>C</code> can be specified.</td>
</tr>
<tr>
<td></td>
<td>» Up to two letters of <code>d</code>, <code>h</code>, <code>k</code>, <code>m</code>, and <code>s</code> can be specified.</td>
</tr>
<tr>
<td></td>
<td>» Up to three letters of <code>D</code> can be specified.</td>
</tr>
<tr>
<td><strong>Number/Text</strong></td>
<td>If the count of pattern letters is 3 or greater, use the Text rules above. Otherwise use the Number rules above.</td>
</tr>
<tr>
<td><strong>Fraction</strong></td>
<td>Specifying any number of fractional seconds (<code>S</code>) will accept between 1 and 6 fractional seconds digits.</td>
</tr>
<tr>
<td><strong>Year</strong></td>
<td>The count of letters determines the minimum field width below which padding is used.</td>
</tr>
<tr>
<td></td>
<td>» If the count of letters is two, then a reduced two digit form is used. This will parse using the base value of 2000 to 2099 inclusive.</td>
</tr>
<tr>
<td></td>
<td>» If the count of letters is less than four (but not two), then the sign is only parsed for negative years. Otherwise, the sign is parsed if the pad width is exceeded.</td>
</tr>
<tr>
<td><strong>Zoneld</strong></td>
<td>This specifies the time-zone ID, such as <code>Europe/Paris</code>. If the count of letters is two, then the time-zone ID is used. Any other count of letters results in a syntax error.</td>
</tr>
<tr>
<td><strong>Zone names</strong></td>
<td>This specifies the display name of the time-zone ID. If the count of letters is one, two or three, then the short name is used. If the count of letters is four, then the full name is used. Five or more letters results in a syntax error.</td>
</tr>
<tr>
<td><strong>Offset O</strong></td>
<td>This formats the localized offset based on the number of pattern letters.</td>
</tr>
<tr>
<td></td>
<td>» One letter specifies the short form of the localized offset, which is localized offset text, such as <code>GMT</code>, with hour without leading zero, optional 2-digit minute and second if non-zero, and colon, for example <code>GMT+8</code>.</td>
</tr>
<tr>
<td></td>
<td>» Four letters specifies the full form, which is localized offset text, such as <code>GMT</code>, with 2-digit hour and minute field, optional second field if non-zero, and colon, for example <code>GMT+08:00</code>.</td>
</tr>
<tr>
<td></td>
<td>» Any other count of letters results in a syntax error.</td>
</tr>
<tr>
<td>Pattern Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Offset Z</td>
<td>This formats the offset based on the number of pattern letters.</td>
</tr>
<tr>
<td></td>
<td>One, two or three letters specifies the hour and minute, without a colon, such as +0130. This will be +0000 when the offset is zero.</td>
</tr>
<tr>
<td></td>
<td>Four letters specifies the full form of localized offset, equivalent to four letters of Offset-O. This will be the corresponding localized offset text if the offset is zero.</td>
</tr>
<tr>
<td></td>
<td>Five letters specifies the hour, minute, with optional second if non-zero, with colon. It specifies Z if the offset is zero.</td>
</tr>
<tr>
<td></td>
<td>Six or more letters results in a syntax error.</td>
</tr>
<tr>
<td>Optional section</td>
<td>The optional section markers work exactly like calling DateTimeFormatterBuilder.optionalStart() and DateTimeFormatterBuilder.optionalEnd().</td>
</tr>
<tr>
<td>Pad modifier</td>
<td>Modifies the pattern that immediately follows to be padded with spaces. The pad width is determined by the number of pattern letters. This is the same as calling DateTimeFormatterBuilder.padNext(int).</td>
</tr>
<tr>
<td></td>
<td>For example, ppH specifies the hour-of-day padded on the left with spaces to a width of 2.</td>
</tr>
<tr>
<td></td>
<td>Any unrecognized letter is an error.</td>
</tr>
<tr>
<td></td>
<td>Any non-letter character, other than [,.], { }, # and the single quote is parsed. Despite this, it is recommended to use single quotes around all characters that you want to parse directly to ensure that future changes do not break your application.</td>
</tr>
</tbody>
</table>

**Formatting Examples**

The following table contains a number of examples of date time formats:

<table>
<thead>
<tr>
<th>Date and Time Pattern</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;yyyy.MM.dd G 'at' HH:mm:ss z&quot;</td>
<td>2001.07.04 AD at 12:08:56 PDT</td>
</tr>
<tr>
<td>&quot;EEE, MMM d, ''yy&quot;</td>
<td>Wed, Jul 4, '01</td>
</tr>
<tr>
<td>&quot;h:mm a&quot;</td>
<td>12:08 PM</td>
</tr>
<tr>
<td>Date and Time Pattern</td>
<td>Example</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
</tr>
<tr>
<td>&quot;hh 'o''clock' a, zzzz&quot;</td>
<td>12 o'clock PM, Pacific Daylight Time</td>
</tr>
<tr>
<td>&quot;K:mm a, z&quot;</td>
<td>0:08 PM, PDT</td>
</tr>
<tr>
<td>&quot;yyyyy.MMMM.dd GGG hh:mm aaa&quot;</td>
<td>02001.July.04 AD 12:08 PM</td>
</tr>
<tr>
<td>&quot;EEE, d MMM yyyy HH:mm:ss Z&quot;</td>
<td>Wed, 4 Jul 2001 12:08:56 -0700</td>
</tr>
<tr>
<td>&quot;yyMMddHHmmssZ&quot;</td>
<td>010704120856-0700</td>
</tr>
<tr>
<td>&quot;yyy-MM-dd'T'HH:mm:ss.SSSZ&quot;</td>
<td>2001-07-04T12:08:56.235-0700</td>
</tr>
<tr>
<td>&quot;yyy-MM-dd'T'HH:mm:ss.SSSXXX&quot;</td>
<td>2001-07-04T12:08:56.235-07:00</td>
</tr>
<tr>
<td>&quot;YYYY-'W'ww-e&quot;</td>
<td>2001-W27-3</td>
</tr>
</tbody>
</table>

**Usage Notes**

You can supply a string formatted as a timestamp to `TO_TIME`; it will translate that value into a time. Note, however, that it cannot translate a date string into a date value.

Note that our examples use lowercase year (e.g. `yyyy`) and day (e.g. `dd`) formats; **not** uppercase (e.g. `YYYY` or `DD`) formats. It's easy to get confused, so remember:

The uppercase `YYYY` format, which is not commonly used, specifies week year, which means that you **must** also specify a week-of-week-based-year (`w`) format value.

Similarly, the uppercase `D` format, also uncommon, specifies day-of-the-year, not the day of the month.

**Examples of Using `TO_TIME`**

Here's an example:

```
splice> VALUES TO_TIME('11:44', 'HH:mm');
1
----------
11:44:00
```

The following example that shows two interesting aspects of using `TO_TIME`. 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.
Note that when you specify a zone offset (z) or time zone (Z), Splice Machine interprets the timestamp in its given zone, and then adjusts the time to the time zone setting of the operating system. This means that when a timestamp value specified in the Easter time zone is parsed on an operating system based in the Pacific time zone, it will be adjusted back by three hours; for example:

```
splice> VALUES TO_TIME('2013-06-18T01:03:30.000EDT','yyy-MM-dd''T''HH:mm:ss.SSSz');
1
-----------
22:03:30
```

See Also

- CURRENT_DATE
- DATE data type
- DATE
- DAY
- EXTRACT
- LASTDAY
- MONTH
- MONTH_BETWEEN
- MONTHNAME
- NEXTDAY
- NOW
- QUARTER
- TIME data type
- TIMESTAMP
- TO_CHAR
- TO_DATE
- TO_TIMESTAMP
- WEEK

- Working with Dates in the Developer's Guide
TO_TIMESTAMP

The TO_TIMESTAMP function parses a datetime string according to a formatting specification, and returns a TIMESTAMP value. Note that the input string must represent a timestamp and must match the formatting specification string.

Syntax

TO_TIMESTAMP( timestampStrExpr, formatStr );

timestampStrExpr
A string expression that contains a time that is formatted according to the format string.

formatStr
A string that specifies the format you want applied to the timeStr. See the Date and Time Formats section below for more information about format specification.

Results

The result is always a TIMESTAMP value.

Date and Time Formats

Splice Machine supports date and time format specifications based on the Java DateTimeFormatter class.

Date and time values are used for both parsing input values and pattern types. For example, the format specification yyyy-MM-dd HH:mm:ssZ parses 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.
- Pattern 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 pattern types.
<table>
<thead>
<tr>
<th>Pattern Letter</th>
<th>Meaning</th>
<th>Pattern Type</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Era</td>
<td>Text</td>
<td>AD; Anno Domini; A</td>
</tr>
<tr>
<td>u</td>
<td>Year</td>
<td>Year</td>
<td>2017; 17</td>
</tr>
<tr>
<td>y</td>
<td>Year-of-era</td>
<td>Year</td>
<td>2017; 17</td>
</tr>
<tr>
<td>D</td>
<td>Day-of-year</td>
<td>Number</td>
<td>189; 303</td>
</tr>
<tr>
<td>M/L</td>
<td>Month-of-year</td>
<td>Number/text</td>
<td>6; 06; Jun; June</td>
</tr>
<tr>
<td>d</td>
<td>Day-of-month</td>
<td>Number</td>
<td>12</td>
</tr>
<tr>
<td>Y</td>
<td>Week-based-year</td>
<td>year</td>
<td>2017; 17</td>
</tr>
<tr>
<td>w</td>
<td>Week-of-week-based-year</td>
<td>Number</td>
<td>14; 51</td>
</tr>
<tr>
<td>W</td>
<td>Week-of-month</td>
<td>Number</td>
<td>7</td>
</tr>
<tr>
<td>E</td>
<td>Day-of-week</td>
<td>Text</td>
<td>Mon; Monday; M</td>
</tr>
<tr>
<td>e/c</td>
<td>Localized day-of-week</td>
<td>Number/text</td>
<td>1; 01; Mon; Monday; M</td>
</tr>
<tr>
<td>F</td>
<td>Week-of-month</td>
<td>Number</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>Am-Pm-of-Day</td>
<td>Text</td>
<td>PM</td>
</tr>
<tr>
<td>h</td>
<td>Clock-hour-of-am-pm (1-12)</td>
<td>Number</td>
<td>7; 12</td>
</tr>
<tr>
<td>K</td>
<td>Hour-of-am-pm (0-11)</td>
<td>Number</td>
<td>0; 11</td>
</tr>
<tr>
<td>k</td>
<td>Clock-hour-of-am-pm (1-24)</td>
<td>Number</td>
<td>1; 13</td>
</tr>
<tr>
<td>H</td>
<td>Hour-of-day (0-23)</td>
<td>Number</td>
<td>0; 11; 17</td>
</tr>
<tr>
<td>m</td>
<td>Minute-of-hour</td>
<td>Number</td>
<td>27</td>
</tr>
<tr>
<td>s</td>
<td>Second-of-minute</td>
<td>Number</td>
<td>48</td>
</tr>
<tr>
<td>S</td>
<td>Fraction-of-second</td>
<td>Fraction</td>
<td>978</td>
</tr>
<tr>
<td>A</td>
<td>Milli-of-day</td>
<td>Number</td>
<td>1234</td>
</tr>
<tr>
<td>V</td>
<td>Time-zone ID</td>
<td>Zone-id</td>
<td>America/San_Francisco; Z; -08:30</td>
</tr>
<tr>
<td>Pattern Letter</td>
<td>Meaning</td>
<td>Pattern Type</td>
<td>Example(s)</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------</td>
<td>--------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>z</td>
<td>Time-zone name</td>
<td>Zone-name</td>
<td>Pacific Standard Time; PST</td>
</tr>
<tr>
<td>o</td>
<td>Localized zone-offset</td>
<td>Offset-0</td>
<td>GMT+7; GMT+07:00; UTC-07:00;</td>
</tr>
<tr>
<td>x</td>
<td>Zone-offset 'Z' for zero</td>
<td>Offset-X</td>
<td>z; -07; -0730; -07:30; -073015; -07:30:15;</td>
</tr>
<tr>
<td>x</td>
<td>Zone-offset</td>
<td>Offset-x</td>
<td>+0000; -07; -0730; -07:30; -073015; -07:30:15;</td>
</tr>
<tr>
<td>z</td>
<td>Zone-offset</td>
<td>Offset-Z</td>
<td>+0000; -0800; -08:00;</td>
</tr>
<tr>
<td>p</td>
<td>Pad next</td>
<td>Pad modifier</td>
<td>1</td>
</tr>
<tr>
<td>'</td>
<td>Escape for text</td>
<td>Delimiter</td>
<td></td>
</tr>
<tr>
<td>''</td>
<td>Single quote</td>
<td>Literal</td>
<td>'</td>
</tr>
</tbody>
</table>

**NOTE:** The format characters x, X, and Z all allow the matching of any possible timezone offset, and all cause the same behavior.

**Pattern Types**
How a pattern 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.

<table>
<thead>
<tr>
<th>Pattern Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>The text style is determined based on the number of pattern letters used;</td>
</tr>
<tr>
<td></td>
<td>‣ Less than 4 pattern letters will use the short form.</td>
</tr>
<tr>
<td></td>
<td>‣ Exactly 4 pattern letters will use the full form.</td>
</tr>
<tr>
<td></td>
<td>‣ Exactly 5 pattern letters will use the narrow form. Pattern letters L, c, and q specify the stand-alone form of the text styles.</td>
</tr>
<tr>
<td>Pattern Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Number</td>
<td>If the count of letters is one, then the value is parsed using the minimum number of digits and without padding. Otherwise, the count of digits is used as the width of the field, with the value zero-padded as necessary.</td>
</tr>
<tr>
<td></td>
<td>The following pattern letters have constraints on the count of letters:</td>
</tr>
<tr>
<td></td>
<td>&gt; Only one letter of c and C can be specified.</td>
</tr>
<tr>
<td></td>
<td>&gt; Up to two letters of d, D, h, H, k, K, m, and s can be specified.</td>
</tr>
<tr>
<td></td>
<td>&gt; Up to three letters of D can be specified.</td>
</tr>
<tr>
<td>Number/Text</td>
<td>If the count of pattern letters is 3 or greater, use the Text rules above. Otherwise use the Number rules above.</td>
</tr>
<tr>
<td>Fraction</td>
<td>Specifying any number of fractional seconds ('S') will accept between 1 and 6 fractional seconds digits.</td>
</tr>
<tr>
<td>Year</td>
<td>The count of letters determines the minimum field width below which padding is used.</td>
</tr>
<tr>
<td></td>
<td>&gt; If the count of letters is two, then a reduced two digit form is used. This will parse using the base value of 2000 to 2099 inclusive.</td>
</tr>
<tr>
<td></td>
<td>&gt; If the count of letters is less than four (but not two), then the sign is only parsed for negative years. Otherwise, the sign is parse if the pad width is exceeded.</td>
</tr>
<tr>
<td>Zoneld</td>
<td>This specifies the time-zone ID, such as Europe/Paris. If the count of letters is two, then the time-zone ID is used. Any other count of letters results in a syntax error.</td>
</tr>
<tr>
<td>Zone names</td>
<td>This specifies the display name of the time-zone ID. If the count of letters is one, two or three, then the short name is used. If the count of letters is four, then the full name is used. Five or more letters results in a syntax error.</td>
</tr>
<tr>
<td>Offset O</td>
<td>This formats the localized offset based on the number of pattern letters.</td>
</tr>
<tr>
<td></td>
<td>&gt; One letter specifies the short form of the localized offset, which is localized offset text, such as GMT, with hour without leading zero, optional 2-digit minute and second if non-zero, and colon, for example GMT+8.</td>
</tr>
<tr>
<td></td>
<td>&gt; Four letters specifies the full form, which is localized offset text, such as GMT, with 2-digit hour and minute field, optional second field if non-zero, and colon, for example GMT+08:00.</td>
</tr>
<tr>
<td></td>
<td>&gt; Any other count of letters results in a syntax error.</td>
</tr>
<tr>
<td>Pattern Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Offset Z</td>
<td>This formats the offset based on the number of pattern letters.</td>
</tr>
<tr>
<td></td>
<td>▶ One, two or three letters specifies the hour and minute, without a colon, such as +0130. This will be +0000 when the offset is zero.</td>
</tr>
<tr>
<td></td>
<td>▶ Four letters specifies the full form of localized offset, equivalent to four letters of Offset-O. This will be the corresponding localized offset text if the offset is zero.</td>
</tr>
<tr>
<td></td>
<td>▶ Five letters specifies the hour, minute, with optional second if non-zero, with colon. It specifies Z if the offset is zero.</td>
</tr>
<tr>
<td></td>
<td>▶ Six or more letters results in a syntax error.</td>
</tr>
<tr>
<td>Optional section</td>
<td>The optional section markers work exactly like calling DateTimeFormatterBuilder.optionalStart() and DateTimeFormatterBuilder.optionalEnd().</td>
</tr>
<tr>
<td>Pad modifier</td>
<td>Modifies the pattern that immediately follows to be padded with spaces. The pad width is determined by the number of pattern letters. This is the same as calling DateTimeFormatterBuilder.padNext(int).</td>
</tr>
<tr>
<td></td>
<td>For example, ppH specifies the hour-of-day padded on the left with spaces to a width of 2.</td>
</tr>
<tr>
<td></td>
<td>Any unrecognized letter is an error.</td>
</tr>
<tr>
<td></td>
<td>Any non-letter character, other than [, ], { }, # and the single quote is parsed. Despite this, it is recommended to use single quotes around all characters that you want to parse directly to ensure that future changes do not break your application.</td>
</tr>
</tbody>
</table>

**Formatting Examples**

The following table contains a number of examples of date time formats:

<table>
<thead>
<tr>
<th>Date and Time Pattern</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;yyyy.MM.dd G 'at' HH:mm:ss z&quot;</td>
<td>2001.07.04 AD at 12:08:56 PDT</td>
</tr>
<tr>
<td>&quot;EEE, MMM d, ''yy&quot;</td>
<td>Wed, Jul 4, '01</td>
</tr>
<tr>
<td>&quot;h:mm a&quot;</td>
<td>12:08 PM</td>
</tr>
</tbody>
</table>
### Date and Time Pattern

<table>
<thead>
<tr>
<th>Date and Time Pattern</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;hh 'o' clock' a, zzzz&quot;</td>
<td>12 o'clock PM, Pacific Daylight Time</td>
</tr>
<tr>
<td>&quot;K:mm a, z&quot;</td>
<td>0:08 PM, PDT</td>
</tr>
<tr>
<td>&quot;yyyyy.MMMMM.dd GGG hh:mm aaa&quot;</td>
<td>02001.July.04 AD 12:08 PM</td>
</tr>
<tr>
<td>&quot;EEE, d MMM yyyy HH:mm:ss Z&quot;</td>
<td>Wed, 4 Jul 2001 12:08:56 -0700</td>
</tr>
<tr>
<td>&quot;yyMMddHHmmssZ&quot;</td>
<td>010704120856-0700</td>
</tr>
<tr>
<td>&quot;yyyy-MM-dd'T'HH:mm:ss.SSSZ&quot;</td>
<td>2001-07-04T12:08:56.235-0700</td>
</tr>
<tr>
<td>&quot;yyyy-MM-dd'T'HH:mm:ss.SSSXXX&quot;</td>
<td>2001-07-04T12:08:56.235-07:00</td>
</tr>
<tr>
<td>&quot;YYYY-'W'ww-e&quot;</td>
<td>2001-W27-3</td>
</tr>
</tbody>
</table>

### Usage Notes

You can supply a string formatted as a date or time to `TO_TIMESTAMP`; it will translate that value into a timestamp.

Note that our examples use lowercase year (e.g. `yyyy`) and day (e.g. `dd`) formats; **not** uppercase (e.g. `YYYY` or `DD`) formats. It's easy to get confused, so remember:

The uppercase `YYYY` format, which is not commonly used, specifies week year, which means that you **must** also specify a week-of-week-based-year (`w`) format value.

Similarly, the uppercase `D` format, also uncommon, specifies day-of-the-year, not the day of the month.

### Examples of Using `TO_TIMESTAMP`

Here are examples:

```
spix> VALUES TO_TIMESTAMP( '2019-08-09 14:44:24', 'yyyy-MM-dd HH:mm:ss' );
1
-----------------------------
2019-08-09 14:44:24.0
```

```
spix> VALUES TO_TIMESTAMP('11:44', 'HH:mm');
1
--------
1970-01-01 11:44:00.0
```
The following example that shows two interesting aspects of using `TO_TIMESTAMP`. 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.

Note that when you specify a zone offset (Z) or time zone (z), Splice Machine interprets the timestamp in its given zone, and then adjusts the time to the time zone setting of the operating system. This means that when a timestamp value specified in the Easter time zone is parsed on an operating system based in the Pacific time zone, it will be adjusted back by three hours; for example:

```
splice> VALUES TO_TIMESTAMP('2013-06-18T01:03:30.000EDT','yyyy-MM-dd''T''HH:mm:ss.SSS z');
1
---------
2013-06-17 22:03:30.0
```

**See Also**

- `CURRENT_DATE`
- `DATE data type`
- `DATE`
- `DAY`
- `EXTRACT`
- `LASTDAY`
- `MONTH`
- `MONTH_BETWEEN`
- `MONTHNAME`
- `NEXTDAY`
- `NOW`
- `QUARTER`
- `TIME data type`
- `TIMESTAMP`
- `TO_CHAR`
- `TO_DATE`
- `TO_TIME`
- `WEEK`
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**

```sql
TRIM( [ trimOperands ] trimSource)
```

- **trimOperands**
  ```sql
  {  { 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**
  ```sql
  {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
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

```sql
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);
----------------------
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
UCASE 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 unless you specify the Locale parameter.

Syntax

UCASE or UPPER ( CharacterExpression [, Locale ] )

CharacterExpression

A LONG VARCHAR data type, or any built-in type that is implicitly converted to a string (but not a bit expression).

Locale

Optional. You can specify the locale (language and country) to perform language-specific conversions. If you don’t supply this parameter, the Java Session locale value, which is automatically detected, is used as your locale.

You must specify the locale using the following format: ll_CC, where:

» ll is a lowercase, two-character ISO-639 language name code.

» CC is an uppercase, two-character ISO-3166 country code.

For example, you can use de_DE to specify that German language rules should be used when applying this function, or you could specify en_US to apply American English rules.

Results

If the CharacterExpression evaluates to NULL, this function returns NULL.

In general, the type, length, and maximum length of the returned value are the same as the length and maximum length of the CharacterExpression. However, the data type, length, and maximum length of the result can be different if you’re using a Locale value that differs from the default locale of your database.
This is because a single character may convert into multiple characters, when a location value is involved. For example, if you're applying this function to a CHAR value and the resulting value length exceeds the limits of a CHAR value, the result will be a VARCHAR value. Similarly, converting a VARCHAR value may result in a LONG VARCHAR value, and converting a LONG VARCHAR value may results in a CLOB value.

**Example**

To return the names of players, use the following clause:

```sql
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
```

```sql
splice> SELECT UPPER(DisplayName, 'en_US')
    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
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
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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>TEAM</th>
<th>BIRTHDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jeremy Johnson</td>
<td>Cards</td>
<td>1985-03-15</td>
</tr>
<tr>
<td>Gary Kosovo</td>
<td>Giants</td>
<td>1985-06-12</td>
</tr>
<tr>
<td>Michael Hillson</td>
<td>Cards</td>
<td>1985-11-07</td>
</tr>
<tr>
<td>Mitch Canepa</td>
<td>Cards</td>
<td>1985-11-26</td>
</tr>
<tr>
<td>Edward Erdman</td>
<td>Cards</td>
<td>1985-12-21</td>
</tr>
<tr>
<td>Jeremy Packman</td>
<td>Giants</td>
<td>1989-01-01</td>
</tr>
<tr>
<td>Nathan Nickels</td>
<td>Giants</td>
<td>1989-05-04</td>
</tr>
<tr>
<td>Ken Strater</td>
<td>Cards</td>
<td>1989-07-20</td>
</tr>
<tr>
<td>Marcus Bamburger</td>
<td>Giants</td>
<td>1989-08-01</td>
</tr>
<tr>
<td>George Goomba</td>
<td>Cards</td>
<td>1989-08-08</td>
</tr>
<tr>
<td>Jack Hellman</td>
<td>Cards</td>
<td>1989-08-09</td>
</tr>
<tr>
<td>Elliot Andrews</td>
<td>Giants</td>
<td>1989-08-21</td>
</tr>
<tr>
<td>Henry Socomy</td>
<td>Giants</td>
<td>1989-11-17</td>
</tr>
</tbody>
</table>

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
# Clauses

This section contains the reference documentation for the Splice Machine SQL Clauses, in the following topics:

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINT</td>
<td>Optional clause in <code>ALTER TABLE</code> statements that specifies a rule to which the data must conform.</td>
</tr>
<tr>
<td>EXCEPT</td>
<td>Takes the distinct rows in the results from one a <code>SELECT</code> statement.</td>
</tr>
<tr>
<td>FROM</td>
<td>A clause in a <code>SelectExpression</code> that specifies the tables from which the other clauses of the query can access columns for use in expressions.</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>Part of a <code>SelectExpression</code> that groups a result into subsets that have matching values for one or more columns.</td>
</tr>
<tr>
<td>HAVING</td>
<td>Restricts the results of a <code>GROUP BY</code> clause in a <code>SelectExpression</code>.</td>
</tr>
<tr>
<td>LIMIT n</td>
<td>Limits the number of results returned by a query.</td>
</tr>
<tr>
<td>OVER</td>
<td>Used in window functions to define the window on which the function operates.</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>Allows you to specify the order in which rows appear in the result set.</td>
</tr>
<tr>
<td>RESULT OFFSET and FETCH FIRST</td>
<td>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.</td>
</tr>
<tr>
<td>TOP n</td>
<td>Limits the number of results returned by a query.</td>
</tr>
<tr>
<td>UNION</td>
<td>Combines the result sets from two queries into a single table that contains all matching rows.</td>
</tr>
<tr>
<td>USING</td>
<td>Specifies which columns to test for equality when two tables are joined.</td>
</tr>
<tr>
<td>WHERE</td>
<td>An optional part of a <code>UPDATE</code> statement that lets you select rows based on a Boolean expression.</td>
</tr>
<tr>
<td>WITH</td>
<td>Allows you to name subqueries to make your queries more readable and/or to improve efficiency.</td>
</tr>
</tbody>
</table>
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 constraint 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

```sql
{ NOT NULL | [ CONSTRAINT constraint-Name] { PRIMARY KEY } } 
```

```sql
{ 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 can only add or remove a primary key using `ALTER TABLE` if the table is empty.

**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 can only add or remove a primary key using `ALTER TABLE` if the table is empty.

**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

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.

**NOTE:** At this time, you can only add or remove a primary key using ALTER TABLE if the table is empty.

### 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 <= 3000000.

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.
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:

```sql
CREATE TABLE Test1
INSERT INTO Test1 VALUES
    ('1','A',1,1,1,1,'2017-07-01'),
    ('1','A',1,1,2,2,'2017-07-02'),
    ('3','C',3,1,3,3,'2017-07-03'),
    ('1','A',1,2,1,1,'2017-07-01'),
    ('1','A',1,2,2,2,'2017-07-02'),
    ('3','C',3,2,3,3,'2017-07-03');
```

Example: Query Using Column Names:

```sql
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;
```

<table>
<thead>
<tr>
<th>INDIV_ID</th>
<th>TRACK_SEQ</th>
<th>REC_SEQ_NBR</th>
<th>TRIAD_CB_DT</th>
<th>RANKING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2017-07-01</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2017-07-02</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2017-07-03</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2017-07-01</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2017-07-02</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2017-07-03</td>
<td>1</td>
</tr>
</tbody>
</table>

6 rows selected

Example: Query Using Column Positions:

```sql
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;
```

<table>
<thead>
<tr>
<th>INDIV_ID</th>
<th>TRACK_SEQ</th>
<th>REC_SEQ_NBR</th>
<th>TRIAD_CB_DT</th>
<th>RANKING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2017-07-01</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2017-07-02</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2017-07-03</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2017-07-01</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2017-07-02</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2017-07-03</td>
<td>1</td>
</tr>
</tbody>
</table>

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
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 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:

```sql
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:

```sql
SELECT *, rank() OVER (PARTITION BY "playerID" ORDER BY "G") as rank FROM Batting WHERE rank = 1;
```

Instead, you need to use a subquery:

```sql
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:

<table>
<thead>
<tr>
<th>Frame Clause Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recycled</td>
<td>BETWEEN UNBOUNDED PRECEEDING AND UNBOUNDED FOLLOWING</td>
</tr>
<tr>
<td>Cumulative</td>
<td>BETWEEN UNBOUNDED PRECEEDING AND CURRENT ROW</td>
</tr>
<tr>
<td>Rolling</td>
<td>BETWEEN 2 PRECEEDING AND 2 FOLLOWING</td>
</tr>
</tbody>
</table>

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**

```sql
-- 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**
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.
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.
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 UNION SELECT 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 ALL SELECT id,i1,i2 FROM t2 ORDER BY id,i1,i2;

<table>
<thead>
<tr>
<th>ID</th>
<th>I1</th>
<th>I2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>5</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>6</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>102</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>103</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>104</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Expressions</td>
<td>Overview of expression syntax and rules.</td>
</tr>
<tr>
<td><strong>Boolean</strong> Expressions</td>
<td>Syntax for and examples of <strong>Boolean</strong> expressions.</td>
</tr>
<tr>
<td><strong>CASE</strong> Expression</td>
<td>Syntax for and examples of <strong>CASE</strong> expressions.</td>
</tr>
<tr>
<td>Dynamic Parameters</td>
<td>Description of using dynamic parameters in expressions in prepared statements.</td>
</tr>
<tr>
<td>Expression Precedence</td>
<td>Specifies operator precedence in expressions.</td>
</tr>
<tr>
<td><strong>NEXT VALUE FOR</strong> Expression</td>
<td>Retrieves the next value from a sequence generator.</td>
</tr>
<tr>
<td><strong>SELECT</strong> Expression</td>
<td>Builds a table value based on filtering and projecting values from other tables.</td>
</tr>
<tr>
<td><strong>TABLE</strong> Expression</td>
<td>Specifies a table, view, or function in a <strong>FROM</strong> clause.</td>
</tr>
<tr>
<td><strong>VALUES</strong> Expression</td>
<td>Constructs a row or a table from other values.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Expression Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column reference</td>
<td>A column-Name that references the value of the column made visible to the expression containing the Column reference.</td>
</tr>
<tr>
<td></td>
<td>You must qualify the column-Name by the table name or correlation name if it is ambiguous.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Constant</td>
<td>Most built-in data types typically have constants associated with them (as shown in the Data types section).</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL is an untyped constant representing the unknown value.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Expression Type</strong></td>
<td><strong>Explanation</strong></td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Dynamic parameter</td>
<td>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. 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. Allowed anywhere in an expression where the data type can be easily deduced. See Dynamic parameters.</td>
</tr>
<tr>
<td>CAST expression</td>
<td>Allows you to specify the type of NULL or of a dynamic parameter or convert a value to another type. See <strong>CAST</strong> function.</td>
</tr>
<tr>
<td>Scalar subquery</td>
<td>Subquery that returns a single row with a single column. See <strong>ScalarSubquery</strong>.</td>
</tr>
<tr>
<td>Table subquery</td>
<td>Subquery that returns more than one column and more than one row. See <strong>TableSubquery</strong>. Allowed as a tableExpression in a FROM clause and with EXISTS, IN, and quantified comparisons.</td>
</tr>
<tr>
<td>Conditional expression</td>
<td>A conditional expression chooses an expression to evaluate based on a boolean test. Conditional expressions include the <strong>COALESCE</strong> function.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Expression Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, −, *, /, unary + and − expressions</td>
<td>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. Unary + is a noop (i.e., +4 is the same as 4). Unary − is the same as multiplying the value by -1, effectively changing its sign.</td>
</tr>
<tr>
<td>AVG</td>
<td>AVG function</td>
</tr>
<tr>
<td>SUM</td>
<td>SUM function</td>
</tr>
<tr>
<td>LENGTH</td>
<td>LENGTH function.</td>
</tr>
<tr>
<td>LOWER</td>
<td>LOWER function.</td>
</tr>
<tr>
<td>COUNT</td>
<td>COUNT function, including COUNT(*)</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Expression Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A CHAR or VARCHAR value that uses wildcards.</td>
<td>The wildcards % and _ make a character string a pattern against which the LIKE operator can look for a match.</td>
</tr>
<tr>
<td>Concatenation expression</td>
<td>In a concatenation expression, the concatenation operator,</td>
</tr>
<tr>
<td>Built-in string functions</td>
<td>The built-in string functions act on a String and return a string. See UCASE or UPPER function.</td>
</tr>
</tbody>
</table>
**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.

<table>
<thead>
<tr>
<th>Expression Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURRENT_DATE</td>
<td>Returns the current date. See the <strong>CURRENT_DATE</strong> function.</td>
</tr>
<tr>
<td>CURRENT_TIME</td>
<td>Returns the current time. See the <strong>CURRENT_TIME</strong> function.</td>
</tr>
<tr>
<td>CURRENT_TIMESTAMP</td>
<td>Returns the current timestamp. See the <strong>CURRENT_TIMESTAMP</strong> function.</td>
</tr>
</tbody>
</table>

**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 the **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:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| **AND, OR, NOT**          | { Expression AND Expression  
|                           | | Expression OR Expression  
|                           | | NOT Expression }        |
| **Comparisons**           | Expression  
|                           | { <  
|                           | | =  
|                           | | >  
|                           | | <=  
|                           | | >=  
|                           | | <> }                     |
| **IS NULL, IS NOT NULL**  | Expression IS [ NOT ] NULL                                            |
| **LIKE**                  | CharacterExpression  
|                           | [ NOT ] LIKE CharacterExpression  
|                           | WithWildCard [ ESCAPE 'escapeCharacter' ]                           |
| **BETWEEN**               | Expression [ NOT ] BETWEEN Expression AND Expression }                |
### Boolean Expressions

The following example presents examples of the Boolean operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Explanation and Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND, OR, NOT</strong></td>
<td>Evaluate any operand(s) that are boolean expressions:</td>
</tr>
<tr>
<td></td>
<td>(orig_airport = 'SFO') OR (dest_airport = 'GR U')</td>
</tr>
</tbody>
</table>

**Comparisons**

<, =, >, <=, >=, <> are applicable to all of the built-in types.

```
DATE('1998-02-26') < DATE('1998-03-01')
```

-- returns true

**NOTE:** Splice Machine also accepts the != operator, which is not included in the SQL standard.
<table>
<thead>
<tr>
<th>Operator</th>
<th>Explanation and Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IS NULL, IS NOT NULL</strong></td>
<td>Test whether the result of an expression is null or not.</td>
</tr>
<tr>
<td></td>
<td>WHERE MiddleName IS NULL</td>
</tr>
<tr>
<td><strong>LIKE</strong></td>
<td>Attempts to match a character expression to a character pattern, which is a character string that includes one or more wildcards.</td>
</tr>
<tr>
<td></td>
<td>% matches any number (zero or more) of characters in the corresponding position in first character expression.</td>
</tr>
<tr>
<td></td>
<td>_ matches one character in the corresponding position in the character expression.</td>
</tr>
<tr>
<td></td>
<td>Any other character matches only that character in the corresponding position in the character expression.</td>
</tr>
<tr>
<td></td>
<td>city LIKE 'Sant_'</td>
</tr>
<tr>
<td></td>
<td>To treat % or _ as constant characters, escape the character with an optional escape character, which you specify with the ESCAPE clause.</td>
</tr>
<tr>
<td></td>
<td>SELECT a FROM tabA WHERE a LIKE '%=_' ESCAPE '='</td>
</tr>
</tbody>
</table>

**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.
<table>
<thead>
<tr>
<th>Operator</th>
<th>Explanation and Example</th>
</tr>
</thead>
</table>
| **BETWEEN** | 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 \( \leq \) and \( \geq \) can be applied.  

```sql
WHERE booking_date
   BETWEEN DATE('1998-02-26')
   AND DATE('1998-03-01')
```

**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: \( \text{col} \text{ between } X \text{ and } Y \) is equivalent to \( \text{col} \geq X \text{ and } \text{col} \leq Y \). Which means that the result set will be empty if your second operand is less than your first. |
| **IN** | 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.  

```sql
WHERE booking_date NOT IN
   (SELECT booking_date
       FROM HotelBookings
       WHERE rooms_available = 0)
```
| **EXISTS** | 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.  

```sql
WHERE EXISTS
   (SELECT *
       FROM Flights
       WHERE dest_airport = 'SFO'
       AND orig_airport = 'GRU')
```
<table>
<thead>
<tr>
<th>Operator</th>
<th>Explanation and Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantified comparison</td>
<td>A quantified comparison is a comparison operator (&lt;, =, &gt;, &lt;=, &gt;=, &lt;&gt; ) with ALL or ANY or SOME applied.</td>
</tr>
<tr>
<td></td>
<td>Operates on table subqueries, which can return multiple rows but must return a single column.</td>
</tr>
<tr>
<td></td>
<td>If ALL is used, the comparison must be true for all values returned by the table subquery. If ANY or SOME is used, the comparison must be true for at least one value of the table subquery. ANY and SOME are equivalent.</td>
</tr>
<tr>
<td></td>
<td>WHERE normal_rate &lt; ALL (SELECT budget/550 FROM Groups)</td>
</tr>
</tbody>
</table>

**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.

\[
\text{CASE} \\
\quad \text{WHEN booleanExpression THEN thenExpression} \\
\quad [ \text{WHEN booleanExpression} \\
\quad \quad \text{THEN thenExpression} \ldots \\
\quad \quad \text{ELSE elseExpression} \\
\text{END}
\]

\textit{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

\[
\begin{align*}
\text{-- returns 3} \\
\text{CASE WHEN 1=1 THEN 3 ELSE 4 END;}
\end{align*}
\]

\[
\begin{align*}
\text{-- returns 7} \\
\text{CASE} \\
\quad \text{WHEN 1 = 2 THEN 3} \\
\quad \text{WHEN 4 = 5 THEN 6} \\
\quad \text{ELSE 7} \\
\text{END;}
\end{align*}
\]
**Dynamic Parameters**

You can prepare statements that are allowed to have parameters for which the value is not specified when the statement is prepared 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**

```java
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 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.

  ```sql
  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.

  ```sql
  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.

  ```sql
  ```
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:

```sql
SELECT *
FROM tab1
WHERE ? = (SELECT x FROM tab2)
```

```sql
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.

```sql
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:

```sql
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.

```sql
SELECT LENGTH(?)
```

Qualified comparisons.

```sql
? = SOME (SELECT 1 FROM t)
```

-- is valid. Dynamic parameter assumed to be INTEGER type

```sql
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**

```sql
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:**

```
{ *
  * | { table-Name | 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:

```sql
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
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:

```sql
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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>TEAM</th>
<th>BIRTHDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jeremy Johnson</td>
<td>Cards</td>
<td>1985-03-15</td>
</tr>
<tr>
<td>Gary Kosovo</td>
<td>Giants</td>
<td>1985-06-12</td>
</tr>
<tr>
<td>Michael Hillson</td>
<td>Cards</td>
<td>1985-11-07</td>
</tr>
<tr>
<td>Mitch Canepa</td>
<td>Cards</td>
<td>1985-11-26</td>
</tr>
<tr>
<td>Edward Erdman</td>
<td>Cards</td>
<td>1985-12-21</td>
</tr>
<tr>
<td>Jeremy Packman</td>
<td>Giants</td>
<td>1989-01-01</td>
</tr>
<tr>
<td>Nathan Nickels</td>
<td>Giants</td>
<td>1989-05-04</td>
</tr>
<tr>
<td>Ken Straiter</td>
<td>Cards</td>
<td>1989-07-20</td>
</tr>
<tr>
<td>Marcus Bamburger</td>
<td>Giants</td>
<td>1989-08-01</td>
</tr>
<tr>
<td>George Goomba</td>
<td>Cards</td>
<td>1989-08-08</td>
</tr>
<tr>
<td>Jack Hellman</td>
<td>Cards</td>
<td>1989-08-09</td>
</tr>
<tr>
<td>Elliot Andrews</td>
<td>Giants</td>
<td>1989-08-21</td>
</tr>
<tr>
<td>Henry Socomy</td>
<td>Giants</td>
<td>1989-11-17</td>
</tr>
</tbody>
</table>

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:

```sql
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:

```sql
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

```plaintext
{  
  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

```sql
-- 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

```java
{ VALUES ( Value {, Value }* )
   [ , ( Value {, Value }* ) ]* |
   VALUES Value [ , Value ]*
}
[ ORDER BY clause ]
[ result offset clause ]
[ fetch first clause ]
```

Value

<table>
<thead>
<tr>
<th>Expression</th>
<th>DEFAULT</th>
</tr>
</thead>
</table>

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:
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:

```sql
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
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:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>The CHAR data type provides for fixed-length storage of strings.</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>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.</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>The VARCHAR data type provides for variable-length storage of strings.</td>
</tr>
</tbody>
</table>

Date and Time Data Types

These are the date and time data types:
The **DATE** data type provides for storage of a year-month-day in the range supported by `java.sql.Date`.

The **TIME** data type provides for storage of a time-of-day value.

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:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BLOB</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>CLOB</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>TEXT</strong></td>
<td>Exactly the same as CLOB.</td>
</tr>
</tbody>
</table>

### Numeric Data Types

These are the numeric data types:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BIGINT</strong></td>
<td>The BIGINT data type provides 8 bytes of storage for integer values.</td>
</tr>
</tbody>
</table>
| **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. |
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOUBLE</td>
<td>The DOUBLE data type provides 8-byte storage for numbers using IEEE floating-point notation. DOUBLE PRECISION can be used synonymously with DOUBLE.</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>The DOUBLE PRECISION data type provides 8-byte storage for numbers using IEEE floating-point notation. DOUBLE can be used synonymously with DOUBLE PRECISION.</td>
</tr>
<tr>
<td>FLOAT</td>
<td>The FLOAT data type is an alias for either a REAL or DOUBLE PRECISION data type, depending on the precision you specify.</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER provides 4 bytes of storage for integer values.</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>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.</td>
</tr>
<tr>
<td>REAL</td>
<td>The REAL data type provides 4 bytes of storage for numbers using IEEE floating-point notation.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>The SMALLINT data type provides 2 bytes of storage.</td>
</tr>
<tr>
<td>TINYINT</td>
<td>The TINYINT data type provides 1 byte of storage.</td>
</tr>
</tbody>
</table>

**Other Data Types**

These are the other data types:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>Provides 1 byte of storage for logical values.</td>
</tr>
</tbody>
</table>
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
**About Numeric Data Types**

This section contains the reference documentation for the numeric data types built into Splice Machine SQL:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>The BIGINT data type provides 8 bytes of storage for integer values.</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>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.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>The DOUBLE data type provides 8-byte storage for numbers using IEEE floating-point notation. DOUBLE PRECISION can be used synonymously with DOUBLE.</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>The DOUBLE PRECISION data type provides 8-byte storage for numbers using IEEE floating-point notation. DOUBLE can be used synonymously with DOUBLE PRECISION.</td>
</tr>
<tr>
<td>FLOAT</td>
<td>The FLOAT data type is an alias for either a REAL or DOUBLE PRECISION data type, depending on the precision you specify.</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER provides 4 bytes of storage for integer values.</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>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.</td>
</tr>
<tr>
<td>REAL</td>
<td>The REAL data type provides 4 bytes of storage for numbers using IEEE floating-point notation.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>The SMALLINT data type provides 2 bytes of storage.</td>
</tr>
</tbody>
</table>
Using Numeric Types

Numeric types include the following types, which provide storage of varying sizes:

<table>
<thead>
<tr>
<th>Numeric Type</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integer numerics</strong></td>
<td></td>
</tr>
<tr>
<td><em>TINYINT</em></td>
<td><em>(1 byte)</em></td>
</tr>
<tr>
<td><em>SMALLINT</em></td>
<td><em>(2 bytes)</em></td>
</tr>
<tr>
<td><em>INTEGER</em></td>
<td><em>(4 bytes)</em></td>
</tr>
<tr>
<td><em>BIGINT</em></td>
<td><em>(8 bytes)</em></td>
</tr>
<tr>
<td><strong>Floating-point (also called approximate) numerics</strong></td>
<td></td>
</tr>
<tr>
<td><em>REAL</em></td>
<td><em>(4 bytes)</em></td>
</tr>
<tr>
<td><em>DOUBLE PRECISION</em></td>
<td><em>(8 bytes)</em></td>
</tr>
<tr>
<td><em>REAL</em></td>
<td></td>
</tr>
<tr>
<td><strong>Exact numerics</strong></td>
<td></td>
</tr>
<tr>
<td><em>DECIMAL</em></td>
<td><em>(storage based on precision)</em></td>
</tr>
<tr>
<td><em>DECIMAL</em></td>
<td></td>
</tr>
</tbody>
</table>

Numeric Type Promotion in Expressions

The following table shows the result type of numeric expressions based on the mix of numeric data types in the expressions.

<table>
<thead>
<tr>
<th>Largest Type That Appears in Expression</th>
<th>Resulting Type of Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>DOUBLE PRECISION</em></td>
<td><em>DOUBLE PRECISION</em></td>
</tr>
<tr>
<td><em>REAL</em></td>
<td><em>DOUBLE PRECISION</em></td>
</tr>
<tr>
<td><em>DECIMAL</em></td>
<td><em>DECIMAL</em></td>
</tr>
</tbody>
</table>
### About Numeric Data Types

<table>
<thead>
<tr>
<th>Largest Type That Appears in Expression</th>
<th>Resulting Type of Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
</tr>
<tr>
<td>INTEGER</td>
<td>BIGINT</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>BIGINT</td>
</tr>
<tr>
<td>TINYINT</td>
<td>BIGINT</td>
</tr>
</tbody>
</table>

For example:

```sql
-- returns a double precision value
VALUES 1 + 1.0e0;

-- returns a decimal value
VALUES 1 + 1.0;

-- returns a bigint value
VALUES CAST (1 AS INT) + CAST (1 AS INT);
```

### Storing Numeric Values

An attempt to put a floating-point type of a larger storage size into a location of a smaller size fails only if the value cannot be stored in the smaller-size location. For example:

```sql
create table mytable (r REAL, d DOUBLE PRECISION);

0 rows inserted/updated/deleted

INSERT INTO mytable (r, d) values (3.4028236E38, 3.4028235E38);

ERROR X0X41: The number '3.4028236E38' is outside the range for the data type REAL.
```

You can store a floating point type in an INTEGER column; the fractional part of the number is truncated. For example:

```sql
INSERT INTO mytable(integer_column) values (1.09e0);

1 row inserted/updated/deleted

SELECT integer_column
FROM mytable;

---------------
1
```

Integer types can always be placed successfully in approximate numeric values, although with the possible loss of some precision.

Integers can be stored in decimals if the DECIMAL precision is large enough for the value. For example:
### Scale for Decimal Arithmetic

SQL statements can involve arithmetic expressions that use decimal data types of different *precisions* (the total number of digits, both to the left and to the right of the decimal point) and *scales* (the number of digits of the fractional component).

The precision and scale of the resulting decimal type depend on the precision and scale of the operands.

Given an arithmetic expression that involves two decimal operands:

- \( lp \): stands for the precision of the left operand
- \( rp \): stands for the precision of the right operand
- \( ls \): stands for the scale of the left operand
- \( rs \): stands for the scale of the right operand

Use the following formulas to determine the scale of the resulting data type for the following kinds of arithmetical expressions:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiplication</td>
<td>( ls + rs )</td>
</tr>
<tr>
<td>division</td>
<td>( 38 - lp + ls - rs )</td>
</tr>
<tr>
<td>AVG( )</td>
<td>( \max(\max(ls, rs), 4) )</td>
</tr>
<tr>
<td>all others</td>
<td>( \max(ls, rs) )</td>
</tr>
</tbody>
</table>

For example, the scale of the resulting data type of the following expression is 34:
Use the following formulas to determine the precision of the resulting data type for the following kinds of arithmetical expressions:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiplication</td>
<td>$lp + rp$</td>
</tr>
<tr>
<td>addition</td>
<td>$2 \times (p - s) + s$</td>
</tr>
<tr>
<td>division</td>
<td>$lp - ls + rp + \max(ls + rp - rs + 1, 4)$</td>
</tr>
<tr>
<td>all others</td>
<td>$\max(lp - ls, rp - rs) + 1 + \max(ls, rs)$</td>
</tr>
</tbody>
</table>

**Rounding of Numeric Expressions**

When the result of a numeric expression needs to be stored into a column that does not have enough significant figures to hold all of the result’s digits, Splice Machine rounds the result value. For example, note that the result below is rounded up due to the precision of the table column $a$:

```
splice> CREATE TABLE t1 (a DECIMAL(10,2));
0 rows inserted/updated/deleted
splice> INSERT INTO t1 VALUES (0.555);
1 row inserted/updated/deleted
splice> SELECT a FROM t1;
A
---------
0.56
1 row selected
```

**See Also**

- Data Type Compatability
- **BIGINT** data type
- **BLOB** data type
- **BOOLEAN** data type
- **CHAR** data type
- **CLOB** data type
- **DATE** data type
About Numeric Data Types

- **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
- **TINYINT** 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, BLOB strings are not associated with a code page. In addition, BLOB strings 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 database
            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);
```
The CHAR data type provides for fixed-length storage of strings.

Syntax

```java
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
);
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 `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( largeCol CLOB(65535));
```
The `DATE` data type provides for storage of a year-month-day in the range supported by `java.sql.Date`.

Unlike `TIME` and `TIMESTAMP` values, `DATE` values are stored as fixed strings, and are not affected by the time zone of the server to which you are connected.

**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 38. If not specified, the default precision is 5.

Prior to version 2.8 of Splice Machine, precision was limited to 31. If you're using an earlier version of either Splice Machine, the Splice Machine JDBC driver, or the Splice Machine `sqlshell.sh` command line interpreter, precision is limited to 31.

*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

```plaintext
DOUBLE
```

or, alternately

```plaintext
DOUBLE PRECISION
```

### Usage Notes

Here are several usage notes for the **DOUBLE**/**DOUBLE PRECISION** data type:

- The following range limitations apply:

<table>
<thead>
<tr>
<th>Limit type</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smallest <strong>DOUBLE</strong> value</td>
<td>-1.79769E+308</td>
</tr>
<tr>
<td>Largest <strong>DOUBLE</strong> value</td>
<td>1.79769E+308</td>
</tr>
<tr>
<td>Smallest positive <strong>DOUBLE</strong> value</td>
<td>2.225E-307</td>
</tr>
<tr>
<td>Largest negative <strong>DOUBLE</strong> value</td>
<td>-2.225E-307</td>
</tr>
</tbody>
</table>

**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.

```plaintext
-- 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**

```plaintext
java.lang.Double
```

**JDBC Metadata Type (java.sql.Types)**

```plaintext
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**

```plaintext
DOUBLE PRECISION
```

or, alternately

```plaintext
DOUBLE
```

**Usage Notes**

Here are several usage notes for the **DOUBLE/DDOUBLE** **PRECISION** data type:

- The following range limitations apply:

<table>
<thead>
<tr>
<th>Limit type</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smallest DOUBLE value</td>
<td>-1.79769E+308</td>
</tr>
<tr>
<td>Largest DOUBLE value</td>
<td>1.79769E+308</td>
</tr>
<tr>
<td>Smallest positive DOUBLE value</td>
<td>2.225E-307</td>
</tr>
<tr>
<td>Largest negative DOUBLE value</td>
<td>-2.225E-307</td>
</tr>
</tbody>
</table>

**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

```sql
NUMERIC [(precision [, scale ])]
```

**precision**
- Must be between 1 and 38. If not specified, the default precision is 5.
- Prior to version 2.8 of Splice Machine, precision was limited to 31. If you're using an earlier version of either Splice Machine, the Splice Machine JDBC driver, or the Splice Machine sqlshell.sh command line interpreter, precision is limited to 31.

**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:
    ```sql
    -- 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 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 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**

```java
REAL
```

**Corresponding Compile-time Java Type**

```java
java.lang.Float
```

**JDBC Metadata Type (java.sql.Types)**

```java
REAL
```

**Limitations**

**REAL value ranges:**

<table>
<thead>
<tr>
<th>Limit type</th>
<th>Limit value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smallest REAL value</td>
<td>-3.402E+38</td>
</tr>
<tr>
<td>Largest REAL value</td>
<td>3.402E+38</td>
</tr>
<tr>
<td>Smallest positive REAL value</td>
<td>1.175E-37</td>
</tr>
<tr>
<td>Largest negative REAL value</td>
<td>-1.175E-37</td>
</tr>
</tbody>
</table>

**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 underflows.

Numeric floating point constants are limited to a length of 30 characters.

```sql
-- 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\(^3\) (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 \(<\), \(\leq\), \(>\), \(\geq\) 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**

```sql
CREATE TABLE myTable( txtCol TEXT(65535));
```
TIME

The TIME data type provides for storage of a time-of-day value. Splice Machine displays TIME and TIMESTAMP values using the current time zone for the server to which you are connected. DATE values, however, are stored as fixed strings, and are not affected by the time zone of the server to which you are connected.

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 time zone for the server to which you are connected. 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**

```sql
VALUES TIME('15:09:02');
VALUES '15:09:02';
```

**See Also**

- `CURRENT_DATE`
- `CURRENT_TIME`
- `CURRENT_TIMESTAMP`
- `DATE` type
- `DATE` function
- `DAY`
- `EXTRACT`
- `LASTDAY`
- `MONTH`
- `MONTH_BETWEEN`
- `MONTHNAME`
- `NEXTDAY`
- `NOW`
- `QUARTER`
- `TIME` function
- `TIMESTAMP` type
- `TIMESTAMP` function
- `TO_CHAR`
- `TO_DATE`
- `WEEK`
- [Working with Dates](#)
TIMESTAMP

The TIMESTAMP data type represents a combined date and time value. Splice Machine stores timestamp values with up to 6 fractional second digits in databases, and allows you to specify literal timestamp values with up to 9 fractional second digits.

Splice Machine displays TIME and TIMESTAMP values using the current time zone for the server to which you are connected. DATE values, however, are stored as fixed strings, and are not affected by the time zone of the server to which you are connected.

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:

<table>
<thead>
<tr>
<th>Pattern Letter</th>
<th>Description</th>
<th>Format(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>year</td>
<td>yy or yyyy</td>
</tr>
<tr>
<td>M</td>
<td>month</td>
<td>MM</td>
</tr>
<tr>
<td>d</td>
<td>day in month</td>
<td>dd</td>
</tr>
<tr>
<td>a</td>
<td>Am/pm marker (AM or PM)</td>
<td>a or aa</td>
</tr>
<tr>
<td>h</td>
<td>hour (0-12)</td>
<td>hh</td>
</tr>
</tbody>
</table>
### Pattern Letter

<table>
<thead>
<tr>
<th>Pattern Letter</th>
<th>Description</th>
<th>Format(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>hour (0-23)</td>
<td>HH</td>
</tr>
<tr>
<td>m</td>
<td>minute in hour</td>
<td>mm</td>
</tr>
<tr>
<td>s</td>
<td>seconds</td>
<td>ss</td>
</tr>
<tr>
<td>S</td>
<td>tenths of seconds</td>
<td>SSS (up to 6 decimal digits: SSSSSS)</td>
</tr>
<tr>
<td>z</td>
<td>time zone text</td>
<td>e.g. Pacific Standard time</td>
</tr>
<tr>
<td>Z</td>
<td>time zone, time offset</td>
<td>e.g. -0800</td>
</tr>
</tbody>
</table>

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, does not allow for time zone specification, and does not include the AM/PM marker.

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:

<table>
<thead>
<tr>
<th>Timestamp value</th>
<th>Format Pattern</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-03-23 09:45:00</td>
<td>yyyy-MM-dd HH:mm:ss</td>
<td>This is the default pattern.</td>
</tr>
<tr>
<td>2013-03-23 19:45:00.98-05</td>
<td>yyyy-MM-dd HH:mm:ss.SSZ</td>
<td>This pattern allows up to 2 decimal digits of seconds, and requires a time zone specification.</td>
</tr>
<tr>
<td>2013-03-23 09:45:00-07</td>
<td>yyyy-MM-dd HH:mm:ssZ</td>
<td>This patterns requires a time zone specification, but does not allow for decimal digits of seconds.</td>
</tr>
<tr>
<td>2013-03-23 19:45:00.98-0530</td>
<td>yyyy-MM-dd HH:mm:ss.SSZ</td>
<td>This pattern allows up to 2 decimal digits of seconds, and requires a time zone specification.</td>
</tr>
<tr>
<td>2013-03-23 19:45:00.123</td>
<td>yyyy-MM-dd HH:mm:ss.SSS</td>
<td>This pattern allows up to 3 decimal digits of seconds, but does not allow a time zone specification. Note that if your data specifies more than 3 decimal digits of seconds, an error occurs.</td>
</tr>
</tbody>
</table>
### 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 time zone for the server to which you are connected. If there is an ambiguity, the built-in formats shown above take precedence.

**TimeStamp values range from '01 Jan 0001 00:00:00 GMT' to '31 Dec 9999 23:59:59 GMT'.**

### See Also

- [CURRENT_DATE](#)
- [CURRENT_TIME](#)
- [CURRENT_TIMESTAMP](#)
- [DATE type](#)
- [DATE function](#)
- [DAY](#)
- [EXTRACT](#)
- [LASTDAY](#)
- [MONTH](#)
- [MONTH_BETWEEN](#)
- [MONTHNAME](#)
- [NEXTDAY](#)
- [NOW](#)
- [QUARTER](#)
- [TIME type](#)
- [TIME function](#)
- [TIMESTAMP function](#)
- [TO_CHAR](#)
» TO_DATE
» WEEK
» Working with Dates
**TINYINT**

The TINYINT data type provides 1 byte of storage.

**Syntax**

TINYINT

**Corresponding Compile-time Java Type**

java.lang.Byte

**JDBC Metadata Type (java.sql.Types)**

TINYINT

**Usage Notes**

Here are several usage notes for the SMALLINT data type:

- The minimum value is -128 (java.lang.Byte.MIN_VALUE).
- The maximum value is 127 (java.lang.Byte.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.
**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.
Identifiers

This section contains the reference documentation for the Splice Machine SQL Identifiers, in these topics:

- This page provides an introduction to SQLIdentifiers.
- 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 SQLIdentifiers 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 SQLIdentifiers

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):

```sql
CREATE TABLE Baseball.Coaches( ID INT NOT NULL );
```

This view name is stored in system catalogs as PITCHINGCOACHES, since it is not quoted:

```sql
CREATE VIEW PitchingCoaches(RECEIVED) AS VALUES 1;
```

Whereas this view name is quoted, and thus is stored as PitchingCoaches in the system catalog:

```sql
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:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization Identifier</td>
<td>An Authorization Identifier is an SQLIdentifier 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 schemaName is SPLICE. 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.</td>
</tr>
<tr>
<td>column-Name</td>
<td>A column-Name is a SQLIdentifier that can be unqualified simple-column-Names or can be qualified with a table-name or correlation-name. See the Column Name Notes section below for information about when a column-Name can or cannot be qualified.</td>
</tr>
<tr>
<td>column-Position</td>
<td>A column-Position is an integer value that specifies the ordinal position value of the column. The first column is column 1.</td>
</tr>
<tr>
<td>column-Name-or-Position</td>
<td>A column-Name-or-Position is either a column-Name or column-Position value.</td>
</tr>
<tr>
<td>constraint-Name</td>
<td>A constraint-Name is a simple SQLIdentifier used to name constraints. You cannot qualify a constraint-Name.</td>
</tr>
<tr>
<td>Identifier Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>correlation-Name</td>
<td>A correlation-Name is a simple SQLIdentifier used in a FROM clause as a new name or alias for that table.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>index-Name</td>
<td>An index-Name is an SQLIdentifier that can be qualified with a schemaName.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>new-Table-Name</td>
<td>A new-Table-Name is a simple SQLIdentifier that is used when renaming a table with the RENAME TABLE statement.</td>
</tr>
<tr>
<td></td>
<td>You cannot qualify a new table name with a schema name, because the table already exists in a specific schema.</td>
</tr>
<tr>
<td>RoleName</td>
<td>A RoleName is a simple SQLIdentifier used to name roles in your database.</td>
</tr>
<tr>
<td></td>
<td>You cannot qualify a role name.</td>
</tr>
<tr>
<td>schemaName</td>
<td>A schemaName is used when qualifying the names of dictionary objects such as tables and indexes.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Note that you must always qualify references to system tables with the SYS. prefix, e.g. SYS.SYSROLES.</td>
</tr>
<tr>
<td>simple-column-Name</td>
<td>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.</td>
</tr>
<tr>
<td>synonym-Name</td>
<td>A synonym-Name is an SQLIdentifier used for synonyms.</td>
</tr>
<tr>
<td></td>
<td>You can optionally be qualify a synonym-Name with a schemaName. If you do not use a qualifying schema name, the default schema is assumed.</td>
</tr>
</tbody>
</table>
A `table-Name` is an `SQLIdentifier` used 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.

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.

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

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’s `correlation-Name` in a `TABLE` expression

### Correlation Name Notes

You cannot use a correlation name for columns that are listed in the `FOR UPDATE` list of a `SELECT`. For example, in the following:

```sql
SELECT Average AS corrCol1, Homeruns AS corrCol2, Strikeouts FROM Batting FOR UPDATE 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:

```sql
CREATE VIEW PitchingCoaches(RECEIVED) AS VALUES 1;
```

is stored in system catalogs as PITCHINGCOACHES, since it is not quoted.

Whereas this view name:

```sql
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:

```sql
CREATE TABLE Coaches( ID INT NOT NULL );
```

And here's an example of a table name (Coaches) qualified by a schema name (Baseball):

```sql
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 sSELECT.
- 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.
Join Operations

This section contains the reference documentation for the Splice Machine SQL Join Operations, in the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About Join Operations</td>
<td>Overview of joins.</td>
</tr>
<tr>
<td>CROSS JOIN</td>
<td>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.</td>
</tr>
<tr>
<td>INNER JOIN</td>
<td>Selects all rows from both tables as long as there is a match between the columns in both tables.</td>
</tr>
<tr>
<td>LEFT OUTER JOIN</td>
<td>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.</td>
</tr>
<tr>
<td>NATURAL JOIN</td>
<td>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.</td>
</tr>
<tr>
<td>RIGHT OUTER JOIN</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Join Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INNER JOIN</td>
<td>Specifies a join between two tables with an explicit join clause.</td>
</tr>
<tr>
<td>LEFT OUTER JOIN</td>
<td>Specifies a join between two tables with an explicit join clause, preserving unmatched rows from the first table.</td>
</tr>
<tr>
<td>RIGHT OUTER JOIN</td>
<td>Specifies a join between two tables with an explicit join clause, preserving unmatched rows from the second table.</td>
</tr>
<tr>
<td>CROSS JOIN</td>
<td>Specifies a join that produces the Cartesian product of two tables. It has no explicit join clause.</td>
</tr>
<tr>
<td>NATURAL JOIN</td>
<td>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.</td>
</tr>
</tbody>
</table>

NOTE: Splice Machine does not currently support NATURAL SELF JOIN operations.

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**

```sql
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:

```sql
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 NULL 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

```sql
-- 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:
splicex> SELECT *
    FROM COUNTRIES
    NATURAL JOIN CITIES;

splicex> 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:

splicex> 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

```
splice> SELECT COUNTRIES.COUNTRY, CITIES.CITY_NAME
  FROM CITIES  RIGHT OUTER JOIN COUNTRIES
  ON CITIES.COUNTRY_ISO_CODE = COUNTRIES.COUNTRY_ISO_CODE;

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';

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:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>Creates a virtual table based on existing tables or constants built into tables.</td>
</tr>
<tr>
<td>Scalar Subquery</td>
<td>A subquery that returns a single row with a single column.</td>
</tr>
<tr>
<td>Table Subquery</td>
<td>A subquery that returns multiple rows.</td>
</tr>
</tbody>
</table>

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**

```sql
{
    ( 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:

```sql
SELECT *
FROM ORG;
```
Here's a SELECT with a subquery:

```sql
SELECT *
FROM (SELECT CLASS_CODE FROM CL_SCHED) AS CS;
```

Here's a SELECT with a subquery:

```sql
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:

```sql
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:

```sql
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
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:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER TABLE</td>
<td>Add, deletes, or modifies columns in an existing table.</td>
</tr>
<tr>
<td>GRANT</td>
<td>Gives privileges to specific user(s) or role(s) to perform actions on database objects.</td>
</tr>
<tr>
<td>RENAME COLUMN</td>
<td>Renames a column in a table.</td>
</tr>
<tr>
<td>RENAME INDEX</td>
<td>Renames an index in the current schema.</td>
</tr>
<tr>
<td>RENAME TABLE</td>
<td>Renames an existing table in a schema.</td>
</tr>
<tr>
<td>REVOKE</td>
<td>Revokes privileges for specific user(s) or role(s) to perform actions on database objects.</td>
</tr>
<tr>
<td>TRUNCATE TABLE</td>
<td>Resets a table to its initial empty state.</td>
</tr>
</tbody>
</table>

Data Definition (DDL) - CREATE Statements

These are the create statements:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE EXTERNAL TABLE</td>
<td>Allows you to query data stored in a flat file as if that data were stored in a Splice Machine table.</td>
</tr>
<tr>
<td>CREATE FUNCTION</td>
<td>Creates Java functions that you can then use in expressions.</td>
</tr>
<tr>
<td>Statement</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CREATE INDEX</td>
<td>Creates an index on a table.</td>
</tr>
<tr>
<td>CREATE PROCEDURE</td>
<td>Creates Java stored procedures, which you can then call using the Call Procedure statement.</td>
</tr>
<tr>
<td>CREATE ROLE</td>
<td>Creates SQL roles.</td>
</tr>
<tr>
<td>CREATE SCHEMA</td>
<td>Creates a schema.</td>
</tr>
<tr>
<td>CREATE SEQUENCE</td>
<td>Creates a sequence generator, which is a mechanism for generating exact numeric values, one at a time.</td>
</tr>
<tr>
<td>CREATE SYNONYM</td>
<td>Creates a synonym, which can provide an alternate name for a table or a view.</td>
</tr>
<tr>
<td>CREATE TABLE</td>
<td>Creates a new table.</td>
</tr>
<tr>
<td>CREATE TEMPORARY TABLE</td>
<td>Defines a temporary table for the current connection.</td>
</tr>
<tr>
<td>CREATE TRIGGER</td>
<td>Creates a trigger, which defines a set of actions that are executed when a database event occurs on a specified table.</td>
</tr>
<tr>
<td>CREATE VIEW</td>
<td>Creates a view, which is a virtual table formed by a query.</td>
</tr>
<tr>
<td>DECLARE GLOBAL TEMPORARY TABLE</td>
<td>Defines a temporary table for the current connection.</td>
</tr>
</tbody>
</table>

**Data Definition (DDL) - DROP Statements**

These are the drop statements:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP FUNCTION</td>
<td>Drops a function from a database.</td>
</tr>
<tr>
<td>DROP INDEX</td>
<td>Drops an index from a database.</td>
</tr>
<tr>
<td>DROP PROCEDURE</td>
<td>Drops a procedure from a database.</td>
</tr>
<tr>
<td>DROP ROLE</td>
<td>Drops a role from a database.</td>
</tr>
<tr>
<td>DROP SCHEMA</td>
<td>Drops a schema from a database.</td>
</tr>
<tr>
<td>DROP SEQUENCE</td>
<td>Drops a sequence from a database.</td>
</tr>
</tbody>
</table>
### Data Manipulation (DML) Statements

These are the data manipulation statements:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL PROCEDURE</td>
<td>Calls a stored procedure.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Deletes records from a table.</td>
</tr>
<tr>
<td>INSERT</td>
<td>Inserts records into a table.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Selects records.</td>
</tr>
<tr>
<td>UPDATE TABLE</td>
<td>Updates values in a table.</td>
</tr>
</tbody>
</table>

### Session Control Statements

These are the session control statements:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET ROLE</td>
<td>Sets the current role for the current SQL context of a session.</td>
</tr>
<tr>
<td>SET SCHEMA</td>
<td>Sets the default schema for a connection's session.</td>
</tr>
</tbody>
</table>
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 **can only** use **ALTER TABLE** to add or drop a primary key if the table is empty.

**Syntax**

```
ALTER TABLE table-Name
{
  ADD COLUMN column-definition |
  ADD CONSTRAINT clause |
  DROP [ COLUMN ] column-name |
  DROP { constraint-name |
         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-name**

An SQLIdentifier specifying the name of the column that you want to drop.

**constraint-name**

An SQLIdentifier specifying the name of the constraint that you want to drop.

If the constraint is unnamed, you can specify the generated **CONSTRAINTNAME ID** that is stored in the **SYS.SYSCONSTRAINTS** table as a delimited value. You can find the constraint ID by joining **SYS.SYSCONSTRAINTS** with **SYS.SYSTABLES**.
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:

```sql
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:

```sql
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**:

```sql
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.

**Dropping constraints**

`ALTER TABLE DROP CONSTRAINT` allows you to drop a constraint from a table. You can specify the constraint by its name, or if the constraint is unnamed, you can specify the generated `CONSTRAINTNAME` ID that is stored in the `SYS.SYSCONSTRAINTS` table as a delimited value. You can find the constraint ID by joining `SYS.SYSCONSTRAINTS` with `SYS.SYSTABLES`.

Example 8: Dropping a Foreign Key Constraint at the end of this topic shows dropping an unnamed foreign key constraint.

**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**
- **Example 2: Altering Columns**
- **Example 3: Dropping a column**
- **Example 4: Changing Varchar Column Width**
- **Example 5: Changing Increment Value**
- **Example 6: Adding a Primary Key**
- **Example 7: Adding a Foreign Key After Table Creation**
- **Example 8: Adding a Foreign Key in Table With Invalid Data**
- **Example 9: Dropping a Foreign Key Constraint**

**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;

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>TYPE_NAME</th>
<th>DEC&amp;</th>
<th>NUM&amp;</th>
<th>COLUM&amp;</th>
<th>COLUMN_DEF</th>
<th>CHAR_OCTE&amp;</th>
<th>IS_NULL&amp;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>INTEGER</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>PLAYERNAME</td>
<td>VARCHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>32</td>
<td>NULL</td>
<td>64</td>
<td>YES</td>
</tr>
<tr>
<td>POSITION</td>
<td>CHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>2</td>
<td>NULL</td>
<td>4</td>
<td>YES</td>
</tr>
<tr>
<td>OLDTEAM</td>
<td>VARCHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>32</td>
<td>NULL</td>
<td>64</td>
<td>YES</td>
</tr>
<tr>
<td>NEWTEAM</td>
<td>VARCHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>32</td>
<td>'Giants'</td>
<td>64</td>
<td>YES</td>
</tr>
<tr>
<td>UPDATED</td>
<td>TIMESTAMP</td>
<td>9</td>
<td>10</td>
<td>29</td>
<td>NULL</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>TRADEDATE</td>
<td>DATE</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>YES</td>
</tr>
<tr>
<td>YEARS</td>
<td>INTEGER</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>3</td>
<td>NULL</td>
<td>YES</td>
</tr>
</tbody>
</table>

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;

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>TYPE_NAME</th>
<th>DEC&amp;</th>
<th>NUM&amp;</th>
<th>COLUM&amp;</th>
<th>COLUMN_DEF</th>
<th>CHAR_OCTE&amp;</th>
<th>IS_NULL&amp;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>INTEGER</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>PLAYERNAME</td>
<td>VARCHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>32</td>
<td>NULL</td>
<td>64</td>
<td>YES</td>
</tr>
<tr>
<td>POSITION</td>
<td>CHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>2</td>
<td>NULL</td>
<td>4</td>
<td>YES</td>
</tr>
<tr>
<td>OLDTEAM</td>
<td>VARCHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>32</td>
<td>NULL</td>
<td>64</td>
<td>YES</td>
</tr>
<tr>
<td>NEWTEAM</td>
<td>VARCHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>32</td>
<td>NULL</td>
<td>64</td>
<td>YES</td>
</tr>
<tr>
<td>UPDATED</td>
<td>TIMESTAMP</td>
<td>9</td>
<td>10</td>
<td>29</td>
<td>NULL</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>TRADEDATE</td>
<td>DATE</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>YES</td>
</tr>
</tbody>
</table>

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;

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>TYPE_NAME</th>
<th>DEC</th>
<th>NUM</th>
<th>COLUM</th>
<th>COLUMN_DEF</th>
<th>CHAR_OCTET</th>
<th>IS_NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>INTEGER</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>PLAYERNAME</td>
<td>VARCHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>40</td>
<td>NULL</td>
<td>80</td>
<td>YES</td>
</tr>
<tr>
<td>POSITION</td>
<td>CHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>2</td>
<td>NULL</td>
<td>4</td>
<td>YES</td>
</tr>
<tr>
<td>OLDTEAM</td>
<td>VARCHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>32</td>
<td>NULL</td>
<td>64</td>
<td>YES</td>
</tr>
<tr>
<td>NEWTEAM</td>
<td>VARCHAR</td>
<td>NULL</td>
<td>NULL</td>
<td>32</td>
<td>NULL</td>
<td>64</td>
<td>YES</td>
</tr>
<tr>
<td>UPDATED</td>
<td>TIMESTAMP</td>
<td>9</td>
<td>10</td>
<td>29</td>
<td>NULL</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>TRADEDATE</td>
<td>DATE</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>NULL</td>
<td>NULL</td>
<td>YES</td>
</tr>
</tbody>
</table>

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;

<table>
<thead>
<tr>
<th>NEWID</th>
<th>PLAYERNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Greinke</td>
</tr>
<tr>
<td>11</td>
<td>Cespedes</td>
</tr>
</tbody>
</table>

2 rows selected

**Example 6: Adding a Primary Key**
This example shows adding a primary key to an empty table.

In this release, you **can only** use `ALTER TABLE` to add or drop a primary key if the table is empty.
Example 7: Adding a Foreign Key After Table Creation
This example shows adding a foreign key into a table that already contains data, and then attempting an invalid insertion to demonstrate that the new constraint is active:

```
CREATE TABLE orders (id INTEGER NOT NULL PRIMARY KEY, name VARCHAR(30));
CREATE TABLE orderlines (lineitemid INTEGER NOT NULL PRIMARY KEY, orderid INTEGER, total DOUBLE);

INSERT INTO orders VALUES (1,'test1');
INSERT INTO orders VALUES (2, 'test2');
INSERT INTO orders VALUES (3, 'test3');

INSERT INTO orderlines VALUES (1,1,12.50);
INSERT INTO orderlines VALUES (2,1,12.50);

ALTER TABLE orderlines ADD CONSTRAINT FK_Order FOREIGN KEY (orderid) REFERENCES Orders(id);
```

This insertion fails because of the newly added foreign key constraint:
```
splice> INSERT INTO ORDERLINES VALUES (3,5,12.50);
ERROR 23503: Operation on table 'ORDERLINES' caused a violation of foreign key constraint 'FK_ORDER' for key (ORDERID). The statement has been rolled back.
```

Example 8: Adding a Foreign Key in Table With Invalid Data
This example shows an attempt to add a foreign key into a table that already contains data that does not conform to the new constraint:

```
CREATE TABLE pktest (i int);
0 rows inserted/updated/deleted
splice> alter table pktest add primary key (i);
0 rows inserted/updated/deleted
splice> show primarykeys from pktest;
TABLE_NAME            |COLUMN_NAME                   |KEY_SEQ   |PK_NAME
---------------------|-----------------------------|----------|-----------------------------
PKTEST               |I                            |1         |SQL19030617
1640400              |                            |          |
```
CREATE TABLE orders3 (id INTEGER NOT NULL PRIMARY KEY, name VARCHAR(30));
CREATE TABLE orderlines3 (lineitemid INTEGER NOT NULL PRIMARY KEY, orderid INTEGER, total DOUBLE);

INSERT INTO orders3 VALUES (1, 'test1');
INSERT INTO orders3 VALUES (2, 'test2');
INSERT INTO orders3 VALUES (3, 'test3');

INSERT INTO orderlines3 VALUES (1, 1, 12.50);
INSERT INTO orderlines3 VALUES (2, 1, 12.50);
INSERT INTO orderlines3 VALUES (3, 5, 12.50);

# Creating this foreign key fails because the table contains non-conformant data:
splice> ALTER TABLE orderlines3 ADD CONSTRAINT FK_Order3 FOREIGN KEY (orderid) REFERENCES Orders3(id);
ERROR X0Y45: Foreign key constraint 'FK_ORDER3' cannot be added to or enabled on table "SPLICE"."ORDERLINES3" because one or more foreign keys do not have matching referenced keys.

**Example 9: Dropping a Foreign Key Constraint**

This example shows an attempt to add a foreign key into a table that already contains data that does not conform to the new constraint, and then after dropping the constraint. Since the constraint is unnamed, we can specify its ID, which you can find by joining `SYS.SYSCONSTRAINTS` with `SYS.SYSTABLES`. In this example, an earlier error message shows us the ID, so we don't need to look it up:
-- create original table
splice> CREATE TABLE t1 (c1 NUMERIC PRIMARY KEY);
0 rows inserted/updated/deleted

-- create secondary table
CREATE TABLE t2 (  
c1 NUMERIC PRIMARY KEY,  
c2 NUMERIC REFERENCES t1(c1) );
0 rows inserted/updated/deleted

-- try (but fail) to enter a row because of fk constraint:
splice> insert into t2 values (1,1);
ERROR 23503: Operation on table 'T2' caused a violation of foreign key constraint 'SQL180830233242711' for key (C2). The statement has been rolled back.

-- enter record correctly:
splice> insert into t1 values 1;
1 row inserted/updated/deleted
splice> insert into t2 values (1,1);
1 row inserted/updated/deleted

-- now get rid of the fk constraint:
splice> alter table t2 drop constraint SQL180830233242711;
0 rows inserted/updated/deleted

-- now we can insert rows that don't reference an fk:
splice> insert into t2 values (2,2);
1 row inserted/updated/deleted

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

\[
\text{CALL procedure-Name} \ ( \\
\text{[ expression [, expression]* ]})
\]

\textit{procedure-Name}  \\
The name of the procedure that you are calling.

\textit{expression(s)}  \\
Arguments passed to the procedure.

Example

The following example depends on a fictionalized java class. For functional examples of using \texttt{CREATE PROCEDURE}, please see the \texttt{Using Functions and Stored Procedures} section in our \textit{Developer's Guide}.

\begin{verbatim}
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(? , ?, ?); 
\end{verbatim}

See Also

» CREATE PROCEDURE statement
## column-definition

<table>
<thead>
<tr>
<th>Simple-column-Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ DataType ]</td>
</tr>
<tr>
<td>[ Column-level-constraint ]*</td>
</tr>
<tr>
<td>[ [ WITH ] DEFAULT DefaultConstantExpression</td>
</tr>
<tr>
<td>generated-column-spec</td>
</tr>
<tr>
<td>generation-clause</td>
</tr>
<tr>
<td>]</td>
</tr>
<tr>
<td>[ Column-level-constraint ]*</td>
</tr>
</tbody>
</table>

### 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
  [ MERGE SCHEMA ]
}
```

* `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 either AVRO or TEXTFILE `file-formats`; doing so generates an error.

* `column-definition` A column definition.

  The maximum number of columns allowed in a table is 131072.
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.

MERGE SCHEMA

This optional setting tells Splice Machine to infer the table schema from all data files, rather than the default behavior of inferring the schema from one data file; it is useful when the external data schema evolves. Using this setting can be very expensive performance-wise.

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.

Our current implementation of AVRO external tables does not support the use of DECIMAL data values. This restriction will be eliminated in a future release.

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.

External Table PARQUET Single File Example
This example creates an external table for the PARQUET data files in the `users/myName/myParquetData` directory, inferring the table schema from one data file in that directory:

```sql
splice> CREATE EXTERNAL TABLE myParquetTable
       (col1 INT, col2 VARCHAR(24))
       PARTITIONED BY (col1)
       STORED AS PARQUET
       LOCATION '/users/myName/myParquetData';
0 rows inserted/updated/deleted
```

External Table PARQUET Multiple Files Example
This example creates an external table for the PARQUET data files in the `users/myName/myParquetData` directory, inferring the table schema from all of the data files in that directory:

```sql
splice> CREATE EXTERNAL TABLE myParquetTable
       (col1 INT, col2 VARCHAR(24))
       PARTITIONED BY (col1)
       STORED AS PARQUET
       LOCATION '/users/myName/myParquetData'
MERGE SCHEMA;
0 rows inserted/updated/deleted
```

External Table Compressed PARQUET Example
This example creates an external table for a PARQUET file that was compressed with Snappy compression:
## External Table AVRO Example

This example creates an external table for an AVRO file:

```sql
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
```

## External Table ORC (on S3) Example

This example creates an external table for an ORC file stored in an AWS S3 bucket and inserts data into it:

```sql
splice> CREATE EXTERNAL TABLE myOrcTable
    (col1 INT, col2 VARCHAR(24))
    PARTITIONED BY (col1)
    STORED AS ORC
    LOCATION 's3a://myOrcData/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;
<table>
<thead>
<tr>
<th>COL1</th>
<th>COL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Three</td>
</tr>
<tr>
<td>2</td>
<td>Two</td>
</tr>
<tr>
<td>1</td>
<td>One</td>
</tr>
</tbody>
</table>
```

**NOTE:** See the [Configuring an S3 Bucket for Splice Machine Access](#) topic for information about accessing data on S3.

## External Table Plain Text Example

This example creates an external table for a plain text file stored on HDFS:
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 'hdfs:///tmp/myTables/myTextFile';
0 rows inserted/updated/deleted

See Also

» CREATE TABLE
» DROP TABLE
» REFRESH EXTERNAL TABLE
» Foreign Keys
» Triggers
**CREATE FUNCTION**

The CREATE FUNCTION statement allows you to create Java or Python functions, which you can then use in expressions.

Creating functions in Python is currently a **Beta Release** feature; it will become generally available in a future release.

For details on how Splice Machine matches functions to Java methods, see Argument matching.

**Syntax**

```
CREATE FUNCTION functionName (  
    [  functionParameter ]  
    [,  functionParameter]  ]  *  
)  
RETURNS returnDataType  [  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**

\[
\text{TABLE}( \text{ColumnElement} [,\text{ColumnElement}]^* )
\]

**ColumnType**

\[
\text{ColumnElement}
\]

A SQL Identifier.

Table functions return TableType results. Currently, only Splice Machine-style table functions are supported, which are functions that return JDBC ResultSets.

You cannot currently write table functions in Python.

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**

\{
\quad \text{LANGUAGE} \ {\ \text{JAVA} \ | \ \text{PYTHON}}
\ | \ {\ \text{EXTERNAL NAME} \ \text{javaMethodName} \ | \ \text{AS} \ ' \ \text{pythonScript}' } \\
\quad \text{DeterministicCharacteristic}
\ | \ \text{PARAMETER STYLE} \ \text{parameterStyle}
\ | \ \text{sqlStatementType}
\ | \ \text{nullInputAction}
\}

The function elements may appear in any order, but each type of element can only appear once.

These function elements are required:

- \text{LANGUAGE}
- \text{EXTERNAL NAME}
- \text{PARAMETER STYLE}

**LANGUAGE**

Specify the language in which your function is written; this must be JAVA or PYTHON.

Creating functions in Python is currently a Beta Release feature; it will become generally available in a future release.
**DeterministicCharacteristic**

<table>
<thead>
<tr>
<th>DETERMINISTIC</th>
<th>NOT DETERMINISTIC</th>
</tr>
</thead>
</table>

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.

**pythonScript**

```def run(scriptArgs): scriptCode```  

This is the Python script, enclosed in single quotes ('). Here are a few important notes about Python scripts in functions, which are described more fully in the Using Functions and Stored Procedures section of our Developer's Guide:

- This feature is currently in Beta release.
- The entire script must be enclosed in single quotes.
- Use double quotes (") around strings within the script; if you must use a single quote within the script, specify it as two single quotes (' ').
- Use spaces instead of tabs within your scripts; the command line processor will convert tabs to a single space in your script, even within a string.
- Write the script under the `run` function.
- The arguments you specify for your script in the CREATE FUNCTION statement should match the order specified in your method definition.

**parameterStyle**

<table>
<thead>
<tr>
<th>JAVA</th>
<th>SPLICE_JDBC_RESULT_SET</th>
</tr>
</thead>
</table>

Only use SPLICE_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 (JAVA or Python)

For more complete examples of using CREATE FUNCTION, please see the Using Functions and Stored Procedures section of our Developer's Guide.

JAVA Example

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

Python Example

Creating functions in Python is currently a Beta Release feature; it will become generally available in a future release.
You can now use this function as follows:

splice> VALUES SPLICE.PYSAMPLE_FUNC('Splice,Machine');

1

Example of declaring a table function (JAVA only)

This example reads data from a mySql database and inserts it into a Splice Machine database.

You cannot currently write table functions in Python.

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:

```java
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.spli
cemachinemachine.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. You can optionally create indexes using bulk HFiles, which can improve index creation performance for large tables.

Syntax

```sql
CREATE [UNIQUE] INDEX indexName
ON tableName (
    simpleColumnName
    [ASC | DESC]
    [, simpleColumnName [ASC | DESC] ] *
)
[ AUTO SPLITKEYS [SAMPLEFRACTION fractionVal ]
| [LOGICAL | PHYSICAL] SPLITKEYS LOCATION filePath
    [colDelimiter] [charDelimiter] [timestampFormat] [dateFormat] [timeFormat]
] [ HFILE hfileLocation ]
[ EXCLUDE ( NULL | DEFAULT ) KEYS ]
```

`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.

`fractionVal`
You can optionally use split keys for index creation, as described below, in the Using Split Keys section; split keys can be computed automatically, or can be specified in a file.

When using automatic (AUTO) splitting, you can specify the sampling fraction in this parameter; this is a decimal value in the range 0 to 1. If you leave this unspecified, the default value stored in the splice.bulkImport.sample.fraction configuration property is used as the sampling fraction.

`filePath`
You can also supply either LOGICAL (primary key) or PHYSICAL (encoded hbase) split keys yourself in a CSV file, instead of using automatic splitting. See the Using Split Keys section for more information.

This parameter value is the path to the CSV file that contains the split key values when using non-automatic splitting.

`colDelimiter`
The character used to separate columns in the CSV file. You don't need to specify this if using the comma (,) character as your delimiter.

For additional information about column delimiters, please see the description in our Importing Data: Input Parameters tutorial page.

**charDelimiter**

The character used to delimit strings in the CSV file. You don't need to specify this if using the double-quote (") character as your delimiter.

For additional information about character delimiters, please see the description in our Importing Data: Input Parameters tutorial page.

**timeStampFormat**

The format of timestamps stored in the CSV file. You don't need to specify this if there are not any 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".

For additional information about timestamp formats, please see the description in our Importing Data: Input Parameters tutorial page.

**dateFormat**

The format of dates stored in the CSV 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".

For additional information about date formats, please see the description in our Importing Data: Input Parameters tutorial page.

**timeFormat**

The format of time values stored in the CSV 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".

For additional information about time formats, please see the description in our Importing Data: Input Parameters tutorial page.

**hFileLocation**

To use the bulk HFile index creation process, you must specify this value, which is the location (full path) in which the temporary HFiles will be created. These files will automatically be deleted after the index creation process completes. If you leave this parameter out, the index will be created without using HFile Bulk loading.

HFile Bulk Loading of indexes 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.

**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:

```sql
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 Split Keys

You can optionally include a file of split keys for the new index; these specify how you want the index to be split over HBase Regions. You can have Splice Machine automatically determine the splits by scanning the data, or you can define the split keys in a CSV file.

The split keys file you provide can specify either `LOGICAL` or `PHYSICAL` keys:

- Logical keys are the primary key column values that you want to define the splits.
- Physical keys are actual split keys for the HBase table, in encoded HBase format.

### Automatic Sampling or Specifying Split Keys

When you specify `AUTO` sampling for index creation, Splice Machine samples the data you’re indexing and determines the best region splits for your index. You can specify the sampling rate to use, as shown below, in [Example 2: Bulk HFile Index Creation with Automatic Split Keys](#).

If you know how your data should be split into regions, you can specify those region split keys in a CSV file, as shown below, in [Example 3: Bulk HFile Index creation with Logical Split Keys](#). Alternatively, if you’re an expert user, you can specify the split keys for the physical HBase table, as shown below, in [Example 4: Bulk HFile Index creation with Physical Split Keys](#).
You can define logical or physical split keys for the index whether or not you're using Bulk HFile loading to create the index. Our Indexing Large Tables tutorial page provides detailed information about using automatic sampling or providing your own region split keys.

### 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.

To learn more about bulk HFile index creation, see Indexing Large Tables tutorial page.

Bulk HFile index creation is related to using Bulk HFiles to import data, which is described in our Importing Data: Bulk HFile Import best practices page.

### Excluding NULL and Default Values

You can include the optional `EXCLUDE` clause to specify that you want to exclude from the index either default values or NULL values for the column. This can be desirable for a large table in which the index column is largely populated with the same (default or NULL) value; excluding these values in the index can mean:

- avoiding a large amount of wasted storage space for the index
- a significant reduction of system resources needed to maintain a very large index

For an index with multiple columns, only rows with NULL or default values on the leading index column are excluded.

Excluding NULL or default values is not applicable for all queries; the Splice Machine optimizer will automatically determine automatically whether or not such an index can be applied in a specific query. The optimizer determines this based on both the cost of using the index, and whether the predicates in the query can guarantee that no rows with NULL or default values could be qualified.

### 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.

**Examples**
This section includes these examples of index creation:

- **Example 1: Simple Index Creation**
- **Example 2: Bulk HFile Index Creation with Automatic Split Keys**
- **Example 3: Bulk HFile Index creation with Logical Split Keys**
- **Example 4: Bulk HFile Index creation with Physical Split Keys**
- **Example 5: Using the EXCLUDE Clause**

**Example 1: Simple Index Creation**
Here's a simple example of creating an index on a table:

```
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
```

**Example 2: Bulk HFile Index Creation with Automatic Split Keys**
This example creates an index using AUTO sampling, with a sampling rate of 0.001:

```
CREATE INDEX l_part_idx ON lineitem(
    l_partkey,
    l_orderkey,
    l_suppkey,
    l_shipdate,
    l_extendedprice,
    l_discount,
    l_quantity,
    l_shipmode,
    l_shipinstruct
) AUTO SPLITKEYS SAMPLE FRACTION 0.001 HFILE LOCATION '/tmp/hfile';
```

**Example 3: Bulk HFile Index creation with Logical Split Keys**
This example creates an index using logical (primary key column values) split keys that are stored in a CSV file:
Example 4: Bulk HFile Index creation with Physical Split Keys {exbulkphysical}
This example creates an index using physical split keys, which are split keys for the HBase table, in encoded HBase format.

CREATE INDEX l_part_idx ON lineitem(
    l_partkey,
    l_orderkey,
    l_suppkey,
    l_shipdate,
    l_extendedprice,
    l_discount,
    l_quantity,
    l_shipmode,
    l_shipinstruct
) PHYSICAL SPLITKEYS LOCATION '/tmp/l_part_idx.txt' HFILE LOCATION '/tmp/hfile';

Example 5: Using the EXCLUDE Clause
This example uses the EXCLUDE DEFAULT KEYS clause, and shows you how the optimizer determines the applicability of the index with that clause.

splice> CREATE TABLE myTable2 (col1 int, col2 int default 5);
0 rows inserted/updated/deleted

splice> CREATE INDEX myIdx2 ON myTable2(col2) EXCLUDE DEFAULT KEYS;
0 rows inserted/updated/deleted

splice> insert into myTable2 values (1,1), (2,2);
2 rows inserted/updated/deleted

splice> insert into myTable2(col1) values 3,4,5;
3 rows inserted/updated/deleted

Now the table contains 5 rows, 3 of which have default values in col2.
As you can see from the generated plan, the optimizer determines that the following query can use the index `myIdx2` because it can tell for sure that the rows with default values on `col2` will not be qualified:

```sql
splice> explain select col2 from mytable2 where col2 < 5;
```

<table>
<thead>
<tr>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

3 rows selected

The optimizer determines that the following query *cannot* use `myIdx2`, because rows with the default value in `col2` could survive the predicate `col2 > 3`:

```sql
splice> select * from myTable2;
```

<table>
<thead>
<tr>
<th>COL1</th>
<th>COL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

5 rows selected

```sql
splice> select col2 from mytable2 where col2 < 5;
```

<table>
<thead>
<tr>
<th>COL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

2 rows selected
splice> explain select col2 from mytable2 where col2 > 3;
Plan
---------------------------------------------------------------------
--------
Cursor(n=3,rows=18,updateMode=READ_ONLY (1),engine=control)
  -> ScrollInsensitive(n=2,totalCost=8.22,outputRows=18,outputHeapSize=18 B,partitions=1)
    -> TableScan[MYTABLE2(1696)](n=1,totalCost=4.04,scannedRows=20,outputRows=18,outputHeapSize=18 B,partitions=1,preds=[(COL2[0:1] > 3)])
3 rows selected
COL2
-------
5
5
5
3 rows selected

Now, if we force the index to be used in the above case, you'll see an error:

splice> explain select col2 from mytable2 --splice-properties index=myIdx2
> where col2 > 3;
ERROR 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.

See Also

» DELETE statement
» DROP INDEX statement
» INSERT statement
» SELECT statement
» UPDATE statement
» Indexing Large Tables
» Importing Data: Bulk HFile Import
CREATE PROCEDURE

The CREATE PROCEDURE statement allows you to create Java or Python procedures, which you can then call using the CALL PROCEDURE statement.

Creating stored procedures in Python is currently a Beta Release feature; it will become generally available in a future release.

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 | PYTHON } |
  { EXTERNAL NAME javaMethodName | AS ' pythonScript ' } |
  DeterministicCharacteristic |
  PARAMETER STYLE parameterStyle |
  DYNAMIC RESULT SETS integer |
  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**

Specify the language in which your procedure is written; this must be JAVA or PYTHON.

⚠️ Creating stored procedures in Python is currently a **Beta Release** feature; it will become generally available in a future release.

**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.

**pythonScript**

```
def run(scriptArgs): scriptCode
```

This is the Python script, enclosed in single quotes (''). Here are a few important notes about Python scripts in stored procedures, which are described more fully in the Using Functions and Stored Procedures section of our Developer's Guide:
This feature is currently in Beta release.

The entire script must be enclosed in single quotes.

Use double quotes (") around strings within the script; if you must use a single quote within the script, specify it as two single quotes (" ").

Use spaces instead of tabs within your scripts; the command line processor will convert tabs to a single space in your script, even within a string.

Write the script under the `run` function.

The arguments you specify for your script in the `CREATE PROCEDURE` statement should match the order specified in your method definition.

### `parameterStyle`

**JAVA**

Stored procedures use 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.

### `DYNAMIC RESULT SETS integer`

Specifies the number of dynamic result sets produced by the procedure.

Currently, Splice Machine only supports 0 or 1 dynamic result sets.

### `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.

### Examples

This section contains two examples of creating procedures: one in JAVA, and another in PYTHON. For functional examples of using `CREATE PROCEDURE`, please see the [Using Functions and Stored Procedures](#) section of our *Developer's Guide*. 
Example of Creating a Stored Procedure in Java
The following example depends on a fictionalized java class.

splice> CREATE PROCEDURE SALES.TOTAL_REVENUE ( 
   IN S_MONTH INTEGER, 
   IN S_YEAR INTEGER, OUT TOTAL DECIMAL(10,2) ) 
PARAMETER STYLE JAVA 
LANGUAGE JAVA 
DYNAMIC RESULT SETS 1 
READS SQL DATA 
EXTERNAL NAME 'com.example.sales.calculateRevenueByMonth'; 
0 rows inserted/updated/deleted

Example of Creating a Stored Procedure in Python
The following example creates a Python stored procedure that executes an SQL statement.

Creating stored procedures in Python is currently a Beta Release feature; it will become generally available in a future release.

splice> CREATE PROCEDURE SPLICE.PYTHON_TEST ( 
   IN limit INT ) 
PARAMETER STYLE JAVA 
LANGUAGE PYTHON 
DYNAMIC RESULT SETS 1 
READS SQL DATA 
AS 'def run(lim, res):
   c = conn.cursor()
   # select alias and javaclassname columns from sys.sysaliases tables
   # return them as a ResultSet
   stmt = "select alias, javaclassname from sys.sysaliases {limit ?}"
   c.executemany(stmt,[lim])
   d = c.description
   result = c.fetchall()
   # construct the ResultSet and fill it into the ResultSet list res
   res[0] = factory.create([d,result])
   conn.commit()
   c.close()
   conn.close()'; 
0 rows inserted/updated/deleted

See Also
- Writing Functions and Stored Procedures
- 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**

```sql
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:

```sql
splice> CREATE SCHEMA FLIGHTS;
0 rows inserted/updated/deleted
```

To create a schema employee-related tables, use the following syntax:

```sql
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
   [ schemaName ]
   [ 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 (TINYINT, 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.

```sql
splice> CREATE SEQUENCE order_id;
0 rows inserted/updated/deleted
```

This example creates a player ID sequence that starts with the integer value 100:

```sql
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.

```sql
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( synonymName
    FOR { viewName | 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.

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} ] *
  ) 
  [ [LOGICAL | PHYSICAL] SPLITKEYS LOCATION filePath] 
  | 
  [ { 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 131072.

**Table-level constraint**

A constraint that applies to the table.

**column-name**

A column definition.

**filePath**

You can optionally specify that you want the new table split among regions by supplying a file of split key values. This capability is typically used when you’re creating a table into which a table backed up with `SYCS_UTIL.SYSCS_BACKUP_TABLE` is being restored. Creating a table with pre-defined splits is much faster than creating a table with one region and then splitting it into many regions.

You can supply either **LOGICAL** (primary key) or **PHYSICAL** (encoded hbase) split keys yourself in a file. See the Using Split Keys section for more information.

This parameter value is the path to the file that contains the split key values when using non-automatic splitting.

**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.

---

**Using Split Keys**

You can optionally include a file of split keys for the new table; you can include split keys when you know how the data that is going to be added to the table should be split into regions. This capability is typically used when you’re creating a table for restoring a table that was previously backed up using the `SYCS_UTIL.SYSCS_BACKUP_TABLE` system procedure.

Creating a table with pre-defined splits is much faster than creating a table with one region and then splitting it into many regions. The split keys file can contain either **LOGICAL** or **PHYSICAL** keys:

- Logical keys are the primary key column values that you want to define the splits.
- Physical keys are actual split keys for the HBase table, in encoded HBase format.
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 NULL values:

```
splicex 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:

```
splicex 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

splicex 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:

```sql
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:

```sql
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:

```sql
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 with SPLIT KEYS

Using Logical Split Keys
This is an example of creating a new table that will be split into regions based on the primary key values in the lineitemKeys.csv file:
Here's what the lineitemKeys.csv file looks like:

```
1424004,7
2384419,4
3244416,6
5295747,4
```

**Using Physical Split Keys**

This is an example of creating a new table that will be split into regions based on the encoded HBase split keys:
CREATE TABLE LINEITEM (  
  L_ORDERKEY INTEGER 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 CHAR(1),  
  L_LINESTATUS CHAR(1),  
  L_SHIPDATE DATE,  
  L_COMMITDATE DATE,  
  L_RECEIPTDATE DATE,  
  L_SHIPINSTRUCT CHAR(25),  
  L_SHIPMODE CHAR(10),  
  L_COMMENT VARCHAR(44),  
  PRIMARY KEY (L_ORDERKEY, L_LINENUMBER)  
) physical splitkeys location '/temp/lineitemKeys.txt';

Here is what the lineitemKeys.txt file looks like:

\xE4\x15\xBA\x84\x00\x87  
\xE4$b#\x00\x84  
\xE41\x81\x80\x00\x86  
\xE4P\xCE\x83\x00\x84

**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
- CONSTRAINT clause
- DROP 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**

```sql
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
- 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:
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.

**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;

<table>
<thead>
<tr>
<th>PLAYER</th>
<th>TEAM</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robert Cohen</td>
<td>Cards</td>
<td>40</td>
</tr>
<tr>
<td>Jason Larrimore</td>
<td>Cards</td>
<td>37</td>
</tr>
<tr>
<td>David Janssen</td>
<td>Cards</td>
<td>36</td>
</tr>
<tr>
<td>Mitch Hassleman</td>
<td>Cards</td>
<td>35</td>
</tr>
<tr>
<td>Mitch Brandon</td>
<td>Cards</td>
<td>35</td>
</tr>
<tr>
<td>Tam Croonster</td>
<td>Cards</td>
<td>34</td>
</tr>
<tr>
<td>Alex Wister</td>
<td>Cards</td>
<td>34</td>
</tr>
<tr>
<td>Yuri Milleton</td>
<td>Cards</td>
<td>33</td>
</tr>
<tr>
<td>Jonathan Pearlman</td>
<td>Cards</td>
<td>33</td>
</tr>
<tr>
<td>Michael Rastono</td>
<td>Cards</td>
<td>32</td>
</tr>
<tr>
<td>Barry Morse</td>
<td>Cards</td>
<td>32</td>
</tr>
<tr>
<td>Carl Vanamos</td>
<td>Cards</td>
<td>32</td>
</tr>
<tr>
<td>Jan Bromley</td>
<td>Cards</td>
<td>31</td>
</tr>
<tr>
<td>Thomas Hillman</td>
<td>Giants</td>
<td>40</td>
</tr>
<tr>
<td>Mark Briste</td>
<td>Giants</td>
<td>38</td>
</tr>
<tr>
<td>Randy Varner</td>
<td>Giants</td>
<td>38</td>
</tr>
<tr>
<td>Jason Lilliput</td>
<td>Giants</td>
<td>38</td>
</tr>
<tr>
<td>Jalen Ardson</td>
<td>Giants</td>
<td>36</td>
</tr>
<tr>
<td>Sam Castleman</td>
<td>Giants</td>
<td>35</td>
</tr>
<tr>
<td>Alex Paramour</td>
<td>Giants</td>
<td>34</td>
</tr>
<tr>
<td>Jack Peepers</td>
<td>Giants</td>
<td>34</td>
</tr>
<tr>
<td>Norman Aikman</td>
<td>Giants</td>
<td>33</td>
</tr>
<tr>
<td>Craig McGawn</td>
<td>Giants</td>
<td>33</td>
</tr>
<tr>
<td>Kameron Fannais</td>
<td>Giants</td>
<td>33</td>
</tr>
<tr>
<td>Jason Martell</td>
<td>Giants</td>
<td>33</td>
</tr>
<tr>
<td>Harry Pennello</td>
<td>Giants</td>
<td>32</td>
</tr>
<tr>
<td>Jason Minman</td>
<td>Giants</td>
<td>32</td>
</tr>
<tr>
<td>Trevor Imhof</td>
<td>Giants</td>
<td>32</td>
</tr>
<tr>
<td>Steve Raster</td>
<td>Giants</td>
<td>32</td>
</tr>
<tr>
<td>Greg Brown</td>
<td>Giants</td>
<td>31</td>
</tr>
<tr>
<td>Alex Darba</td>
<td>Giants</td>
<td>31</td>
</tr>
<tr>
<td>Joseph Arkman</td>
<td>Giants</td>
<td>31</td>
</tr>
<tr>
<td>Tam Lassiter</td>
<td>Giants</td>
<td>31</td>
</tr>
<tr>
<td>Martin Cassman</td>
<td>Giants</td>
<td>31</td>
</tr>
<tr>
<td>Yuri Piamam</td>
<td>Giants</td>
<td>31</td>
</tr>
</tbody>
</table>

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
- 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='/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.

---

**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—a \texttt{UPDATE WHERE CURRENT} statement depends on the statement it references. Removing the statement on which it depends invalidates the \texttt{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;
s 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 BaseballStats 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**

```sql
DROP TRIGGER TriggerName
```

*TriggerName*

The name of the trigger that you want to drop from your database.

**Example**

```sql
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
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.

```sql
[ 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:

```sql
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, included it in the insertion statement. For example:

```sql
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(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 **TINYINT**, **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.

- **TINYINT**
- **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.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Maximum Value</th>
<th>Minimum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TINYINT</strong></td>
<td>127 (java.lang.Byte.MAX_VALUE)</td>
<td>−128 (java.lang.Byte.MIN_VALUE)</td>
</tr>
<tr>
<td><strong>SMALLINT</strong></td>
<td>32767 (java.lang.Short.MAX_VALUE)</td>
<td>−32768 (java.lang.Short.MIN_VALUE)</td>
</tr>
<tr>
<td>Data Type</td>
<td>Maximum Value</td>
<td>Minimum Value</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>INT</td>
<td>2147483647 (java.lang.Integer.MAX_VALUE)</td>
<td>-2147483648 (java.lang.Integer.MIN_VALUE)</td>
</tr>
<tr>
<td>BIGINT</td>
<td>9223372036854775807 (java.lang.Long.MAX_VALUE)</td>
<td>-9223372036854775808 (java.lang.Long.MIN_VALUE)</td>
</tr>
</tbody>
</table>

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), UWORLD 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;
<table>
<thead>
<tr>
<th>WORD</th>
<th>UWORLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>chocolate</td>
<td>CHOCOLATE</td>
</tr>
<tr>
<td>Coca-Cola</td>
<td>COCA-COLA</td>
</tr>
<tr>
<td>hamburger</td>
<td>HAMBURGER</td>
</tr>
<tr>
<td>carrot</td>
<td>CARROT</td>
</tr>
</tbody>
</table>
4 rows selected
splice> select upper(word) from words;
<table>
<thead>
<tr>
<th>UPPER(word)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHOCOLATE</td>
</tr>
<tr>
<td>COCA-COLA</td>
</tr>
<tr>
<td>HAMBURGER</td>
</tr>
<tr>
<td>CARROT</td>
</tr>
</tbody>
</table>
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;
<table>
<thead>
<tr>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>11</td>
</tr>
</tbody>
</table>
```sql
splice> UPDATE T SET COL2 = 100 WHERE COL1 = 1;
1 row inserted/updated/deleted
splice> select * from t;
<table>
<thead>
<tr>
<th>COL1</th>
<th>COL2</th>
<th>COL3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>11</td>
</tr>
</tbody>
</table>
```

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**

```sql
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, as described in these sections:

- **Syntax for Schemas**
- **Syntax for Tables**
- **Syntax for Roles**
- **Syntax for Routines**
- **Syntax for Sequences**
- **Syntax for User-Defined Types**

This topic also contains these sections that help explain the use of the GRANT statement:

- **About Grantees**
- **Privilege Types**
- **Usage Notes**
- **Examples**

There is no explicit mechanism for granting permission to create indexes; a user must have `modify` permission (grant modify schema) on the schema containing a table to have permission to create an index on that table.

### Syntax for Schemas

```sql
GRANT ALL PRIVILEGES | schema-privilege {, schema-privilege )* |
ON SCHEMA schema-Name
TO grantees
```

**SchemaPrivilege**

- DELETE
- INSERT
- MODIFY
- 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.


\subsection*{schema-Name}
The name of the schema to which you are granting access.

\subsection*{grantees}
The user(s) or role(s) to whom you are granting access. See the About Grantees section below for more information.

\section*{NOTES:}

\begin{itemize}
  \item When you drop a schema from your database, all privileges associated with the schema are removed.
  \item Table-level privileges override schema-level privileges.
  \item To have permissions to create an index on a table, a user must have modify privileges on the schema (grant modify schema) that contains the table; there is not explicit mechanism for granting permissions to create indexes.
\end{itemize}

\section*{Syntax for Tables}

\begin{verbatim}
GRANT ALL PRIVILEGES | table-privilege {, table-privilege }* ON [TABLE] { tablename }
    TO grantees

  table-privilege
  \begin{verbatim}
    DELETE
    | INSERT
    | REFERENCES [( column-identifier {, column-identifier}* )]
    | SELECT [( column-identifier {, column-identifier}* )]
    | TRIGGER
    | UPDATE [( column-identifier {, column-identifier}* )]
  \end{verbatim}
\end{verbatim}

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.


**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 Roles

```sql
GRANT roleName [ {, roleName }* ]
   TO grantees
   [ [NOT] AS DEFAULT ]
```

**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.

**[NOT] AS DEFAULT**

When you grant a role to a user, that role is, by default, applied to the user whenever s/he connects to the database. This is the behavior defined by the optional phrase AS DEFAULT.
If you do not want the role granted to the user by default, you must specify NOT AS DEFAULT; this means that the role will not automatically apply to sessions: you must use the SET ROLE statement to apply a NOT AS DEFAULT role in a session.

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.

**Syntax for Routines**

```sql
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 Sequences**

```sql
GRANT USAGE 
ON SEQUENCE sequence-name
TO grantees
```

`sequence-name`

An SQL_Identifier specifying the name of the sequence 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**

```sql
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.

---

**About Grantees**
A grantee can be one or more specific users or groups, 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.

**NOTE:**
When using an LDAP Group name in a `GRANT` or `REVOKE` statement: if the group name contains characters other than alphanumerics or the underscore character (A-Z, a-z, 0-9, _), you must:

- Enclose the group name in double quotes
- Convert all alphabetic characters in the group name to uppercase.

For example, if you are granting rights to an LDAP Group with name `This-is-my-LDAP-Group`, you would use a statement like this:

```
GRANT SELECT ON TABLE Salaries TO "THIS-IS-MY-LDAP-GROUP";
```

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

<table>
<thead>
<tr>
<th>Privilege Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL PRIVILEGES</td>
<td>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.</td>
</tr>
<tr>
<td>DELETE</td>
<td>To grant permission to delete rows from the specified table.</td>
</tr>
<tr>
<td>INSERT</td>
<td>To grant permission to insert rows into the specified table.</td>
</tr>
<tr>
<td>MODIFY</td>
<td>Schema-level privilege that grants permission to modify the schema itself.</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>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.</td>
</tr>
<tr>
<td>SELECT</td>
<td>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 SelectExpression (for example, queries that use COUNT(*)), the user must have at least one column-level SELECT privilege or table-level SELECT privilege.</td>
</tr>
</tbody>
</table>

**NOTE:** Permission to modify the schema does not imply granting of other permissions; use ALL PRIVILEGES to grant all permissions.
### Privilege Types

<table>
<thead>
<tr>
<th>Privilege Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIGGER</td>
<td>To grant permission to create a trigger on the specified table.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>To grant permission to use the <code>WHERE</code> clause, you must have the <code>SELECT</code> privilege on the columns in the row that you want to update.</td>
</tr>
</tbody>
</table>

### 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

This section contains examples for:

- Granting Privileges to Users (#UserPrivs)
- Granting Roles to Users (#UserRoles)
- Granting Privileges to Roles (#RolePrivs)

#### Granting Privileges to Users

To grant the `SELECT` privilege on the schema `SpliceBBall` to the authorization IDs `Bill` and `Joan`, use the following syntax:
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
```

This grants the role to both users for the current session, and also sets the role as a default role whenever one of the users connects to the database. The *as default* behavior is applied by default, or you can specify it explicitly:

```
splice> GRANT purchases_reader_role TO george,maria AS DEFAULT;
0 rows inserted/updated/deleted
```

To grant the role to george only for the current session, use:

```
splice> GRANT purchases_reader_role TO george NOT AS DEFAULT;
0 rows inserted/updated/deleted
```

### A More Extensive ROLE Example

Let’s set up our example. First we’ll use create 4 schemas and 4 roles, and we’ll grant all privileges to each role on its respective schema:
Next we'll create two users so we can demonstrate assigning different roles to different users:

splice> CALL syscs_util.syscs_create_user('user1', 'user1pswd');
Statement executed;
splice> CALL syscs_util.syscs_create_user('user2', 'user2pswd');
Statement executed

Now we'll grant the role test_role1 to all users (the public user), and GRANT specific roles to specific users:

splice> GRANT test_role1 TO public AS DEFAULT;
0 rows inserted/updated/deleted
splice> GRANT test_role2 TO user1 AS DEFAULT;
0 rows inserted/updated/deleted
splice> GRANT test_role3 TO user1;
0 rows inserted/updated/deleted

Now let's CONNECT as user1 and check our role assignments:
As you can see, when user1 connects, s/he is granted:

- **TEST_ROLE1** because it is now granted by default to all users (public).
- **TEST_ROLE2** and **TEST_ROLE3** because they were granted to user1 as a default privilege upon connecting.

Now we'll CONNECT as user2:

Note that user2 is connected with only one role, TEST_ROLE1 because that role has been GRANTed by default to all users (public) and no other roles have been granted to user2.

**Unsetting the AS DEFAULT Role Setting**

If you want to GRANT a role to a user just for the current session, you can use the `NOT AS DEFAULT` syntax.

You can use the same syntax to modify an existing role from DEFAULT to non-DEFAULT:
splicex> GRANT test_role1 TO public not AS DEFAULT;
Statement executed;

As a result, new public connections will no longer be granted the privileges associated with test_role1.

**Granting Privileges to Roles**

To grant the `SELECT` privilege on schema `SpliceBBall` to the role `purchases_reader_role`, use the following syntax:

```
splicex> 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:

```
splicex> 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:
For more information about queries, see Query.

**Examples**

These examples insert records with literal values:

```
splice> INSERT INTO Players
1 row inserted/updated/deleted
splice> INSERT INTO Players
           (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
**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:

```sql
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.

```sql
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.

```
There is no explicit mechanism for revoking permission to create indexes; you must revoke the user's modify permission on the schema (revoke modify schema) containing the table to revoke permission to create an index on that table.
```

### Syntax for SCHEMA

```
REVOKE privilege-type
    ON SCHEMA schema
FROM grantees
```

**privilege-type**

```
DELETE
| INSERT
| MODIFY
| 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.

```

**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**

```sql
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.

`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**

```sql
REVOKE EXECUTE ON { FUNCTION | PROCEDURE }
    {function-name | procedure-name}
    FROM grantees RESTRICT
```

`function-name | procedure-name`
The name of the function or procedure for which you are revoking access.

\textit{grantees}  
The user(s) or role(s) for whom you are revoking access. See the \textit{About Grantees} section below for more information.

\textbf{RESTRICT}  
You \textbf{must} use this clause when revoking access for routines.

The \textbf{RESTRICT} clause specifies that the \texttt{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.

\section*{Syntax for Sequences}

\begin{verbatim}
REVOKE USAGE ON SEQUENCE sequence-name
FROM grantees
\end{verbatim}

\textit{sequence-name}  
The name of the sequence for which you are revoking access.

\textit{grantees}  
The user(s) or role(s) for whom you are revoking access. See the \textit{About Grantees} section below for more information.

\begin{verbatim}
REVOKE USAGE
ON TYPE SQL Identifier
FROM grantees RESTRICT
\end{verbatim}

\textit{[schema-name.] SQL Identifier}  
The user-defined type (UDT) name is composed of an optional \texttt{schemaName} and a \texttt{SQL Identifier}. If a \texttt{schemaName} is not provided, the current schema is the default schema. If a qualified UDT name is specified, the schema name cannot begin with \texttt{SYS}.

\textit{grantees}  
The user(s) or role(s) for whom you are revoking access. See the \textit{About Grantees} section below for more information.

\textbf{RESTRICT}  
You \textbf{must} use this clause when revoking access for user-defined types.

The \textbf{RESTRICT} clause specifies that the \texttt{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

```sql
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 or groups, 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.

**NOTE:**

When using an LDAP Group name in a GRANT or REVOKE statement: if the group name contains characters other than alphanumerics or the underscore character (A-Z, a-z, 0-9, _), you must:

- Enclose the group name in double quotes
- Convert all alphabetic characters in the group name to uppercase.

For example, if you are revoking rights for an LDAP Group with name This-is-my-LDAP-Group, you would use a statement like this:

```sql
REVOKE SELECT ON TABLE Salaries FROM "THIS-IS-MY-LDAP-GROUP";
```

Only the database owner can grant a role to a user or to another role.

Here's the syntax:

```sql
{     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

<table>
<thead>
<tr>
<th>Privilege Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALL PRIVILEGES</strong></td>
<td>To revoke all of the privileges to the user or role for the specified table. You can also revoke one or more table privileges by specifying a privilege-list.</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>To revoke permission to delete rows from the specified table.</td>
</tr>
<tr>
<td><strong>INSERT</strong></td>
<td>To revoke permission to insert rows into the specified table.</td>
</tr>
<tr>
<td><strong>MODIFY</strong></td>
<td>To revoke permission to modify the schema itself.</td>
</tr>
<tr>
<td><strong>REFERENCES</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>SELECT</strong></td>
<td>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 SelectExpression (for example, queries that use COUNT(*)), the user must have at least one column-level SELECT privilege or table-level SELECT privilege.</td>
</tr>
</tbody>
</table>

**NOTE:** Revoking schema modification privilege does not imply revocation of other permissions; use **ALL PRIVILEGES** to revoke all permissions.
<table>
<thead>
<tr>
<th>Privilege Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIGGER</td>
<td>To revoke permission to create a trigger on the specified table.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>To revoke permission to use the <code>WHERE</code> clause, you must have the <code>SELECT</code> privilege on the columns in the row that you want to update.</td>
</tr>
</tbody>
</table>

**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 SYSCOLPERMS 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 SYSCOLPERMS. 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 SELECT privilege for GRANTEE(user2), TABLEID(user1.t1), TYPE(S).
ser1.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:
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 selects 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;

<table>
<thead>
<tr>
<th>DISPLAYNAME</th>
<th>TEAM</th>
<th>BIRTHDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jeremy Johnson</td>
<td>Cards</td>
<td>1985-03-15</td>
</tr>
<tr>
<td>Gary Kosovo</td>
<td>Giants</td>
<td>1985-06-12</td>
</tr>
<tr>
<td>Michael Hillson</td>
<td>Cards</td>
<td>1985-11-07</td>
</tr>
<tr>
<td>Mitch Canepa</td>
<td>Cards</td>
<td>1985-11-26</td>
</tr>
<tr>
<td>Edward Erdman</td>
<td>Cards</td>
<td>1985-12-21</td>
</tr>
<tr>
<td>Jeremy Packman</td>
<td>Giants</td>
<td>1989-01-01</td>
</tr>
<tr>
<td>Nathan Nickels</td>
<td>Giants</td>
<td>1989-05-04</td>
</tr>
<tr>
<td>Ken Strater</td>
<td>Cards</td>
<td>1989-07-20</td>
</tr>
<tr>
<td>Marcus Bamburger</td>
<td>Giants</td>
<td>1989-08-01</td>
</tr>
<tr>
<td>George Goomba</td>
<td>Cards</td>
<td>1989-08-08</td>
</tr>
<tr>
<td>Jack Hellman</td>
<td>Cards</td>
<td>1989-08-09</td>
</tr>
<tr>
<td>Elliot Andrews</td>
<td>Giants</td>
<td>1989-08-21</td>
</tr>
<tr>
<td>Henry Socomy</td>
<td>Giants</td>
<td>1989-11-17</td>
</tr>
</tbody>
</table>

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 INDEX` statement
- `CREATE VIEW` statement
- `DROP INDEX` statement
- `DROP VIEW` statement
- `GRANT` statement
- `ORDER BY` clause
- `FETCH FIRST` clause
- `RESULT OFFSET` clause
**SET ROLE**

The `SET ROLE` statement allows you to add a role for the current SQL context of a session. The new role is unioned with any roles previously associated with the user’s session.

You can add a role only if the current user has been granted the role, or if the role has been granted to `PUBLIC`.

**Notes**

- 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.
- To unset all roles for the current session, use the `SET ROLE NONE` statement.
- Setting a role applies only to the current session; to automatically set roles for users whenever they connect, use the `GRANT ROLE` statement.

**Syntax**

```
SET ROLE { roleName | 'string-constant' | ? | NONE }
```

- `roleName`  
  The role you want added as a role in the current session.
  
  You can specify a `roleName` of `NONE` to disassociate all roles from the current user session.
  
  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 the `GRANT ROLE` 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 set of roles, if any

**NOTE:** You can find the available role names in the `SYS.SYSROLES` system table.

### SQL Example

This example adds the `reader_role` role to the current user's role settings:

```sql
splice> SET ROLE reader_role;
0 rows inserted/updated/deleted
```

### JDBC Example

This example also adds the `reader_role` role to the current user's role settings:

```java
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
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:
Here's an example using a prepared statement:

```java
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
```
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:
This example updates two columns in a table by selecting values from another table:

```
splice> CREATE TABLE table_a (column2 INT, column3 INT);
0 rows inserted/updated/deleted
splice> CREATE TABLE table_b (columnx INT, columny INT);
0 rows inserted/updated/deleted
splice> INSERT INTO table_b VALUES (1,1),(2,2);
2 rows inserted/updated/deleted
splice> INSERT INTO table_a VALUES (1,10),(2,20);
2 rows inserted/updated/deleted
splice> UPDATE table_a SET (column2, column3) = (SELECT columnx, columny FROM table_b AS b WHERE table_a.column2 = b.columnx);
2 rows inserted/updated/deleted
splice> select * from table_a;
+----------+-------+
<table>
<thead>
<tr>
<th>COLUMN2</th>
<th>COLUMN3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
```

**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 TABLE statement
- CONSTRAINT clause
- CREATE TABLE statement
- CREATE TRIGGER
- DROP INDEX statement
- DROP TRIGGER
- WHERE clause
SYCS_UTIL.SYCS_BACKUP_DATABASE

The SYCS_UTIL.SYCS_BACKUP_DATABASE system procedure performs an immediate full or incremental backup of your database to a specified backup directory.

**ENTREPRISE ONLY:** This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product; contact Splice Machine Sales for information.

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**

```
SYCS_UTIL.SYCS_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 backups run as Spark jobs, submitting tasks to copy HFiles. In the past, Splice Machine backups used the Apache Hadoop `distcp` tool to copy the HFile; `distcp` uses MapReduce to copy, which can require significant resources. These requirements can limit file copying parallelism and reduce backup throughput. Splice Machine backups now can run (and do so by default) using a Spark executor to copy the HFiles, which significantly increases backup performance.

You can revert to using `distcp`, which uses a MapReduce job that can run into resource issues. For more information, see the Understanding and Troubleshooting Backups topic.

Backup and Restore Compatibility
Note that you can only use specific restore procedures with specific types of backups. For example, you can use the `RESTORE_TABLE` procedure to restore from a backup created by `BACKUP_TABLE`, but you cannot use `RESTORE_TABLE` to restore from a backup created by `BACKUP_DATABASE`. The following table summarizes backup-restore compatibility:

<table>
<thead>
<tr>
<th>If you backed up with this procedure:</th>
<th>You can restore with these procedures:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYCS_UTIL.SYCS_BACKUP_DATABASE</td>
<td>SYCS_UTIL.SYCS_RESTORE_DATABASE</td>
</tr>
<tr>
<td>SYCS_UTIL.SYCS_BACKUP_SCHEMA</td>
<td>SYCS_UTIL.SYCS_RESTORE_SCHEMA,</td>
</tr>
<tr>
<td></td>
<td>SYCS_UTIL.SYCS_RESTORE_TABLE</td>
</tr>
<tr>
<td>SYCS_UTIL.SYCS_BACKUP_TABLE</td>
<td>SYCS_UTIL.SYCS_RESTORE_TABLE</td>
</tr>
</tbody>
</table>

Usage Notes
Please review these important notes about usage of this system procedure:

- HBase Configuration Options for Incremental Backup
- Temporary Tables and Backups

HBase Configuration Options for Incremental Backup
If you’re performing incremental backups, you *must* add the following options to your `hbase-site.xml` configuration file:
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.

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:

```java
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:

```sql
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:
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:

CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE( './dbBackups', 'full' );
Statement executed.

See Also

» Backing Up and Restoring Databases
» SYSCS_UTIL.SYSCS_BACKUP_SCHEMA
» SYSCS_UTIL.SYSCS_BACKUP_TABLE
» SYSCS_UTIL.SYSCS_DELETE_BACKUP
» SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS
» SYSCS_UTIL.SYSCS_RESTORE_DATABASE
» SYSCS_UTIL.SYSCS_RESTORE_SCHEMA
» SYSCS_UTIL.SYSCS_RESTORE_TABLE
» SYSBACKUP
» SYSBACKUPITEMS
The SYSCS_UTIL.SYSCS_BACKUP_SCHEMA system procedure performs an immediate full backup of the tables and indexes belonging to a schema in your database to a specified backup directory.

**ENTERPRISE ONLY:** This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product; contact Splice Machine Sales for information.

This procedure only works with internal, Splice Machine tables in your database. You can back up an external table using the Hadoop DistCp tool.

### Syntax

```
SYSCS_UTIL.SYSCS_BACKUP_SCHEMA( VARCHAR schemaName,
                                       VARCHAR directory,
                                       VARCHAR type );
```

**schemaName**

The name of the schema you want to back up.

**directory**

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.

**type**

Specifies the type of schema backup that you want performed. Currently, the only valid value is `full`.

### Results

This procedure does not return a result.
**Backup and Restore Compatibility**

Note that you can only use specific restore procedures with specific types of backups. For example, you can use the RESTORE_TABLE procedure to restore from a backup created by BACKUP_TABLE, but you cannot use RESTORE_TABLE to restore from a backup created by BACKUP_DATABASE. The following table summarizes backup-restore compatibility:

<table>
<thead>
<tr>
<th>If you backed up with this procedure:</th>
<th>You can restore with these procedures:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCS_UTIL.SYSCS_BACKUP_DATABASE</td>
<td>SYCS_UTIL.SYSCS_RESTORE_DATABASE</td>
</tr>
<tr>
<td>SYCS_UTIL.SYSCS_BACKUP_SCHEMA</td>
<td>SYCS_UTIL.SYSCS_RESTORE_SCHEMA,</td>
</tr>
<tr>
<td></td>
<td>SYCS_UTIL.SYCS_RESTORE_TABLE</td>
</tr>
<tr>
<td>SYCS_UTIL.SYSCS_BACKUP_TABLE</td>
<td>SYCS_UTIL.SYSCS_RESTORE_TABLE</td>
</tr>
</tbody>
</table>

**Backing Up and Restoring Statistics**

Note that statistics are also backed up and restored as of version 2.7.0.1924 (August 5, 2019) or later of Splice Machine. This means that if you restore a backup created with 2.7.0.1924 or later and the statistics were accurate when the backup was done, you do not need to use analyze to generate fresh statistics for the table immediately after restoring it. If the statistics were not accurate, you can run analyze to refresh them.

If you've restored from a table or schema backup and aren't sure if statistics were restored, you can use the following query to determine if statistics are available, replacing `<mySchemaName>` and `<myTableName>` with the appropriate names:

```
SELECT * FROM SYSVW.SYSTABLESTATISTICS
WHERE schemaname='<mySchemaName>' and tablename='<myTableName>'
```

**Backup Resource Allocation**

Splice Machine backups run as Spark jobs, submitting tasks to copy HFiles. In the past, Splice Machine backups used the Apache Hadoop distcp tool to copy the HFile; distcp uses MapReduce to copy, which can require significant resources. These requirements can limit file copying parallelism and reduce backup throughput. Splice Machine backups now can run (and do so by default) using a Spark executor to copy the HFiles, which significantly increases backup performance.

You can revert to using distcp, which uses a MapReduce job that can run into resource issues. For more information, see the Understanding and Troubleshooting Backups topic.
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 of the TPCH1 schema to a subdirectory of the /backup directory:

```
CallableStatement cs = conn.prepareCall
    ("CALL SYSCS_UTIL.SYSCS_BACKUP_SCHEMA(?,?,?)");
cs.setString(1, 'TPCH1');
cs.setString(2, '/backup');
cs.setString(3, 'full');
cs.execute();
cs.close();
```

SQL Example: Backup, and Restore a Schema
This example shows you how to back up a schema, then restore it, in these steps:

- **Backing Up the Schema**
- **Examining the Backup**
- **Restoring the Backup**

**Backing Up the Schema**
This command line performs a full backup of the TPCH1 schema to the /backup directory on HDFS:

```
splice> CALL SYCS_UTIL.SYSCS_BACKUP_SCHEMA('TPCH1', '/backup', 'full');
Success
--------------
FULL backup to /backup

1 row selected
```

**Examining the Backup**
After the backup completes, you can examine the sys.sysbackup table to find the ID of your new backup:
You can use the ID of your backup job to examine the `sys.sysbackupitems` and verify that the base table and two indexes have been backed up:

```sql
splice> SELECT * FROM sys.sysbackupitems WHERE backup_Id=125953 ;
```

```
BACKUP_ID   |ITEM             |BEGIN_TIMESTAMP           |END_TIMESTAMP
----------------------------------------------------------------------------------------
- 125953      |splice:292000    |2018-10-26 00:12:40.512   |2018-10-26 00:32:14.856
125953      |splice:292033    |2018-10-26 00:12:40.513   |2018-10-26 00:42:48.573
125953      |splice:292017    |2018-10-26 00:12:40.512   |2018-10-26 00:41:25.683
3 rows selected
```

**Restoring the Backup**

You can restore the schema to another schema on the same cluster, or on a different cluster. You can optionally specify that you want the backup validated before it is restored; the validation process checks for inconsistencies and missing files.

This command first validates the backed-up schema, and then restores it to a different (pre-existing) schema named `NEWTPCH1`:

```sql
splice> CALL SYSCS_UTIL.SYSCS_RESTORE_SCHEMA('NEWTPCH1', 'TPCH1', '/backup', 125953, true);
```

Statement executed.

See the reference page for the `SYSCS_UTIL.SYSCS_RESTORE_SCHEMA` system procedure for more information about restoring a backed-up schema.

**See Also**

- `SYSCS_UTIL.SYSCS_RESTORE_TABLE`
- `SYSCS_UTIL.VALIDATE_TABLE_BACKUP`
- `SYSSBACKUP`
- `SYSSBACKUPITEMS`
- `SYSCS_UTIL.SYSCS_BACKUP_DATABASE`
Backing Up and Restoring Databases
SYSCS_UTIL.SYSCS_BACKUP_TABLE

The SYSCS_UTIL.SYSCS_BACKUP_TABLE system procedure performs an immediate full backup of a table in your database to a specified backup directory.

**ENTERPRISE ONLY:** This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product; contact Splice Machine Sales for information.

This procedure only works with internal, Splice Machine tables in your database. You can back up an external table using the Hadoop DistCp tool.

**Syntax**

```sql
SYSCS_UTIL.SYSCS_BACKUP_TABLE( VARCHAR schemaName,
                               VARCHAR tableName,
                               VARCHAR directory,
                               VARCHAR type );
```

*schemaName*

The name of the table’s schema.

*tableName*

The name of the table that you are backing up.

*directory*

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.

*type*

Specifies the type of table backup that you want performed. Currently, the only valid value is `full`.

**Results**

This procedure does not return a result.
Backup Resource Allocation

Splice Machine backups run as Spark jobs, submitting tasks to copy HFiles. In the past, Splice Machine backups used the Apache Hadoop distcp tool to copy the HFile; distcp uses MapReduce to copy, which can require significant resources. These requirements can limit file copying parallelism and reduce backup throughput. Splice Machine backups now can run (and do so by default) using a Spark executor to copy the HFiles, which significantly increases backup performance.

You can revert to using distcp, which uses a MapReduce job that can run into resource issues. For more information, see the Understanding and Troubleshooting Backups topic.

Backup and Restore Compatibility

Note that you can only use specific restore procedures with specific types of backups. For example, you can use the RESTORE_TABLE procedure to restore from a backup created by BACKUP_TABLE, but you cannot use RESTORE_TABLE to restore from a backup created by BACKUP_DATABASE. The following table summarizes backup-restore compatibility:

<table>
<thead>
<tr>
<th>If you backed up with this procedure:</th>
<th>You can restore with these procedures:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCS_UTIL.SYSCS_BACKUP_DATABASE</td>
<td>SYCS_UTIL.SYSCS_RESTORE_DATABASE</td>
</tr>
<tr>
<td>SYSCS_UTIL.SYSCS_BACKUP_SCHEMA</td>
<td>SYCS_UTIL.SYSCS_RESTORE_SCHEMA,</td>
</tr>
<tr>
<td></td>
<td>SYCS_UTIL.SYSCS_RESTORE_TABLE</td>
</tr>
<tr>
<td>SYCS_UTIL.SYSCS_BACKUP_TABLE</td>
<td>SYCS_UTIL.SYSCS_RESTORE_TABLE</td>
</tr>
</tbody>
</table>

Backing Up and Restoring Statistics

Note that statistics are also backed up and restored as of version 2.7.0.1924 (August 5, 2019) or later of Splice Machine. This means that if you restore a backup created with 2.7.0.1924 or later and the statistics were accurate when the backup was done, you do not need to use analyze to generate fresh statistics for the table immediately after restoring it. If the statistics were not accurate, you can run analyze to refresh them.

If you've restored from a table or schema backup and aren't sure if statistics were restored, you can use the following query to determine if statistics are available, replacing <mySchemaName> and <myTableName> with the appropriate names:

```sql
SELECT * FROM SYSVW.SYSTABLESTATISTICS
WHERE schemaname='"<mySchemaName>"' and tablename='"<myTableName>"
```
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 of the TPCH100 LINEITEM table to a subdirectory of the /backup directory:

```java
CallableStatement cs = conn.prepareCall
    ("CALL SYSCS_UTIL.SYSCS_BACKUP_TABLE(?,?,?,?)");
    cs.setString(1, 'TPCH100');
    cs.setString(2, 'LINEITEM');
    cs.setString(3, '/backup');
    cs.setString(4, 'full');
    cs.execute();
    cs.close();
```

SQL Example: Backup, Validate, and Restore a Table
This example shows you how to back up a table, then validate and restore it, in these steps:

- **Backing Up the Table**
- **Examining the Backup**
- **Validating the Backup**
- **Restoring the Backup**

**Backing Up the Table**
This command line performs a full backup of the TPCH100 LINEITEM table to the /backup directory on HDFS:

```sql
splice> CALL SYSCS_UTIL.SYSCS_BACKUP_TABLE('TPCH100', 'LINEITEM', '/backup', 'full');
Success
--------------
FULL backup to /backup
1 row selected
```

**Examining the Backup**
After the backup completes, you can examine the `sys.sysbackup` table to find the ID of your new backup:
You can use the ID of your backup job to examine the `sys.sysbackupitems` and verify that the base table and two indexes have been backed up:

```
splice> SELECT * FROM sys.sysbackupitems WHERE backup_Id=587516417;
```

Validating the Backup

Before restoring the table, you can validate the backup:

```
splice> CALL SYSCS_UTIL.VALIDATE_TABLE_BACKUP( 'TPCH100', 'LINEITEM', '/backup', 587516417);
```

Restoring the Backup

You can restore the table to another table on the same cluster, or on a different cluster.

This command restores the backed-up table to table named `LINEITEM` in the `SPLICE` schema:

```
splice> CALL SYSCS_UTIL.SYSCS_RESTORE_TABLE('SPLICE', 'LINEITEM', 'TPCH100', 'LINEITEM', '/backup', 587516417, false);
```

See the reference page for the `SYSCS_UTIL.SYSCS_RESTORE_TABLE` system procedure for more information about restoring a backed-up table.
See Also

» SYCS_UTIL.SYCS_BACKUP_SCHEMA
» SYCS_UTIL.SYCS_RESTORE_SCHEMA
» SYCS_UTIL.SYCS_RESTORE_TABLE
» SYCS_UTIL.VALIDATE_TABLE_BACKUP
» SYSBACKUP
» SYSBACKUPITEMS
» SYCS_UTIL.SYCS_BACKUP_DATABASE
» Backing Up and Restoring Databases
The SYCS_UTIL.BULK_IMPORT_HFILE system procedure imports data into your Splice Machine database by splitting the table or index file into HFiles and then importing those HFiles. The splitting can be managed automatically by this procedure, or you can pre-split the data before calling SYCS_UTIL.BULK_IMPORT_HFILE.

Unlike our standard SYCS_UTIL.IMPORT_DATA procedure, our bulk HFile procedure does not perform constraint checking while loading your data.

Our [Best Practices: Ingestion] chapter includes an overview and examples of using bulk HFile import.

Syntax

call SYCS_UTIL.BULK_IMPORT_HFILE (  
    schemaName,  
    tableName,  
    insertColumnList | null,  
    fileOrDirectoryName,  
    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've computed pre-splits for your input data, the parameter values that you pass to that procedures must match the values that you pass to this procedure for the same-named parameters.

Parameters

This table includes a brief description of each parameter; additional information is available in the Ingestion Parameter Values topic of our Importing Data tutorial.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaName</td>
<td>The name of the schema of the table into which to import.</td>
<td>SPLICE</td>
</tr>
<tr>
<td>tableName</td>
<td>The name of the table into which to import.</td>
<td>playerTeams</td>
</tr>
<tr>
<td>insertColumnList</td>
<td>The names, in single quotes, of the columns to import. If this is null, all columns are imported.</td>
<td>'ID, TEAM'</td>
</tr>
</tbody>
</table>
| fileOrDirectoryName | 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. 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. See the Configuring an S3 Bucket for Splice Machine Access topic for information about accessing data on S3. | /data/mydata/mytable.csv
's3a://splice-benchmark-data/flat/TPCH/100/region' |
| columnDelimiter   | The character used to separate columns, Specify null if using the comma (,) character as your delimiter. | '|', '	'                                                                     |
| characterDelimiter | The character 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 Java.sql.Timestamp default format, which is: "yyyy-MM-dd HH:mm:ss". | 'yyyy-MM-dd HH:mm:ss.SSS'                                                    |

**NOTE:** The individual column names in the insertColumnList do not need to be double-quoted, even if they contain special characters. However, if you do double-quote any column name, **you must** double-quote all of the column names.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dateFormat</td>
<td>The format of datestamps 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: &quot;yyyy-MM-dd&quot;.</td>
<td>yyyy-MM-dd</td>
</tr>
<tr>
<td>timeFormat</td>
<td>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: &quot;HH:mm:ss&quot;.</td>
<td>HH:mm:ss</td>
</tr>
<tr>
<td>maxBadRecords</td>
<td>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.</td>
<td>25</td>
</tr>
<tr>
<td>badRecordDirectory</td>
<td>The directory in which bad record information is logged. Splice Machine logs information to the <code>&lt;import_file_name&gt;.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>. 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.</td>
<td>'importErrsDir'</td>
</tr>
<tr>
<td>oneLineRecords</td>
<td>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.</td>
<td>true</td>
</tr>
<tr>
<td>charset</td>
<td>The character encoding of the import file. The default value is UTF-8.</td>
<td>null</td>
</tr>
<tr>
<td>bulkImportDirectory</td>
<td>The name of the directory into which the generated HFiles are written prior to being imported into your database. These files will be deleted after the import has finished.</td>
<td>hdfs:///tmp/test_hfile_import/</td>
</tr>
</tbody>
</table>
**Parameter** | **Description** | **Example Value**
--- | --- | ---
skipSampling | The `skipSampling` parameter is a Boolean value that specifies how you want the split keys used for the bulk HFile import to be computed. Set to `false` to have `SYSCS_UTIL.BULK_IMPORT_HFILE` automatically determine splits for you. If `skipSampling` is `true`, you need to use `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX` (recommended) or `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS` (for expert users) to pre-compute splits for your table before calling `SYSCS_UTIL.BULK_IMPORT_HFILE`. If `skipSampling` is `false`, then `SYSCS_UTIL.BULK_IMPORT_HFILE` samples your input data and computes the table splits for you by performing 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.

This allows you more control over the splits, but adds a layer of complexity. You can learn about computing splits for your input data in the Using Bulk HFile Import topic of our Best Practices Guide. | `false` |

**Usage**

Before it generate HFiles, `SYSCS_UTIL.BULK_IMPORT_HFILE` must use `split keys` to determine how to split the data file into multiple HFiles. Splitting the file into evenly-size HFiles yields optimal data loading performance.

You have these choices for determining how the data is split:

» You can call `SYSCS_UTIL.BULK_IMPORT_HFILE` with the `skipSampling` parameter set to `false`; this procedure then samples and analyzes the data in your file and splits the data into temporary HFiles based on that analysis. You’ll find an example in the Best Practices: Bulk Importing Flag Files chapter of our Best Practices Guide.
You can pre-split your data by first creating a CSV file that specifies the split keys to use to perform the pre-splits, and then calling the `SYCS_UTIL.SYCS_SPLIT_TABLE_OR_INDEX` to pre-split your table or index file. You'll also find an example of this in the Best Practices: Bulk Importing Flag Files chapter of our Best Practices Guide. * `SYCS_UTIL.BULK_IMPORT_HFILE` automatically deletes the temporary HFiles after the import process has completed.

### Usage Notes
A few important notes:

- Splice Machine advises you to run a full compaction (with the `SYCS_UTIL.SYCS_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
```

### Results
`SYCS_UTIL.BULK_IMPORT_HFILE` displays a summary of the import process results that looks like this:

<table>
<thead>
<tr>
<th>rowsImported</th>
<th>failedRows</th>
<th>files</th>
<th>dataSize</th>
<th>failedLog</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>0</td>
<td>1</td>
<td>4720</td>
<td>NONE</td>
</tr>
</tbody>
</table>

### Examples
You'll find examples of using this procedure in the Best Practices: Bulk Importing Flat Files topic of our Best Practices Guide.

### See Also
- Best Practices: Ingestion
- Bulk Importing Flat Files
- `SYCS_UTIL.IMPORT_DATA`
- `SYCS_UTIL.MERGE_DATA_FROM_FILE`
» SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE

» SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX
SYSCS_UTIL.SYSCS_CANCEL_BACKUP

The SYSCS_UTIL.SYSCS_CANCEL_BACKUP system procedure cancels an in-progress backup.

Syntax

```sql
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:

```sql
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.CHECK_TABLE

The SYSCS_UTIL.CHECK_TABLE system procedure reports on inconsistencies between a table and its indexes; it reports these categories of problems:

- Unmatched row and index count
- Missing indexes
- Invalid indexes that do not refer to a base table row
- Duplicate indexes that refer to the same base table row

**Syntax**

```sql
SYSCS_UTIL.CHECK_TABLE( VARCHAR schemaName,
                         VARCHAR tableName,
                         VARCHAR indexName,
                         INTEGER level,
                         VARCHAR outputFile)
```

- **schemaName**
  Specifies the schema of the table that you want to check.

- **tableName**
  Specifies the name of the table that you want to check. You can specify `NULL` to check all tables in the named schema.

- **indexName**
  Specifies the name of the index that you want to check. You can specify `NULL` to check all indexes on the named table.

- **level**
  Specifies the level of error checking that you want performed. There are two possible values:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SYSCS_UTIL.CHECK_TABLE counts the number of tables and indexes, and reports an error if they do not match.</td>
</tr>
<tr>
<td>2</td>
<td>SYSCS_UTIL.CHECK_TABLE reports invalid indexes, missing indexes, and duplicate indexes.</td>
</tr>
</tbody>
</table>

- **outputFile**
  The name of a file to which the reported information is written.
Results
This procedure writes a report to an output file; if no inconsistencies are found, the output file is not created. Several example results files are displayed in the Examples section below.

Examples

Example 1: No Inconsistencies Found
In this example, no inconsistencies are found in the specified table:

```
splice> CALL SYSCS_UTIL.CHECK_TABLE('sys', 'systables', null, 2, '/Users/MyName/tmp/checksystables.out');
RESULT
-----------------------------------------
No inconsistencies were found.
1 row selected
```

Example 2: Inconsistencies Reported at Level 1
In this example, the examined table contains some level 1 inconsistencies:

```
splice> CALL SYSCS_UTIL.CHECK_TABLE('SPLICE', 'MyTableA', null, 1, '/Users/MyName/tmp/checkmytableA.1.out');
RESULT
---------------------------------------------------------------
Found inconsistencies. Check /Users/MyName/tmp/checkmytableA.Level1.out for details.
1 row selected
```

The report in /Users/MyName/tmp/checkmytableA.Level1.out displays row counts for the base table MyTableA and the index MyTableAI:

```
MyTableA:
    count = 2
MyTableAI:
    count = 4
```

Example 2: Inconsistencies Reported at Level 2
In this example, we examine the same table and report the inconsistencies at Level 2:
splice> CALL SYSCS_UTIL.CHECK_TABLE('SPLICE', 'MyTableA', null, 2, '/Users/MyName/tmp/checkmytableA.Level2.out');
RESULT
---------------------------------------------------------------
Found inconsistencies. Check /Users/MyName/tmp/checkmytableA.Level2.out for details.
1 row selected

The report in /Users/MyName/tmp/checkmytableA.Level2.out displays three inconsistencies for the base table MyTableA and the index MyTableAI:

AI:

- The following 2 rows from base table SPLICE.MyTableA are not indexed:
  - { 4, 4 }
  - { 5, 5 }
- The following 2 indexes are invalid:
  - { 2 }=>820082
  - { 1 }=>810081
- The following 2 indexes are duplicates:
  - { 7 }=>870087
  - { 8 }=>870087

> The first message indicates that two rows with primary keys (4,4) and (5,5) are not indexed. Note that SYSCS_UTIL.CHECK_TABLE only reports primary key values; if a table does not have a primary key, it reports the row ID instead.

> Two indexes with index column values 2 and 1 do not index any rows: there are no rows with row ID 820082 or 810081 in the base table.

> Two indexes are indexing the same base table row, 87007.
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**

```sql
SYCS_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:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaName</td>
<td>VARCHAR</td>
<td>The name of the schema.</td>
</tr>
<tr>
<td>tableName</td>
<td>VARCHAR</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>partition</td>
<td>VARCHAR</td>
<td>The name of the region on which statistics were collected.</td>
</tr>
<tr>
<td>rowsCollected</td>
<td>INTEGER</td>
<td>The number of rows of statistics that were collected.</td>
</tr>
<tr>
<td>partitionSize</td>
<td>BIGINT</td>
<td>The size of the partition in bytes.</td>
</tr>
</tbody>
</table>

**Usage Notes**
Collecting statistics on a schema can take some time.

**SQL Examples**

```
splice> CALL SYSCS_UTIL.COLLECT_SCHEMA_STATISTICS( 'SPLICE', false );
```

```
<table>
<thead>
<tr>
<th>schemaName</th>
<th>tableName</th>
<th>partition</th>
<th>rowsCollected</th>
<th>partitionSize</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPLICE</td>
<td>PLAYERS</td>
<td>splice:1440..</td>
<td>76</td>
<td>351</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1467393447889.cbc33f4635ade</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPLICE</td>
<td>SALARIES</td>
<td>splice:1456..</td>
<td>76</td>
<td>1420</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1467393749257.7724e0cb12af3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPLICE</td>
<td>BATTING</td>
<td>splice:1472..</td>
<td>44</td>
<td>2257</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1467393754889.b34f5da64c36e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPLICE</td>
<td>PITCHING</td>
<td>splice:1488..</td>
<td>32</td>
<td>21212</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1467393760434.35ee9880e5090</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPLICE</td>
<td>FIELDING</td>
<td>splice:1504..</td>
<td>44</td>
<td>9876</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1467393775949.674b34acdb182</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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*
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**

```sql
SYCS_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 SYCS_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('SPLICE', 'TESTTABLE', null, '1|2', '|', null, null, null, null);)
<table>
<thead>
<tr>
<th>ENCODED_REGION_NAME</th>
<th>START_KEY</th>
<th>END_KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>8ffc80e3f8ac3b180441371319ea90e2</td>
<td>\x81\x00\x81</td>
<td>\x81\x00\x84</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 row selected

splice> CALL SYSCS_UTIL.COMPACT_REGION('SPLICE', 'testTable', NULL, '8ffc80e3f8ac3b180441371319ea90e2');
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('SPLICE', 'TESTTABLE', 'SHIP_INDEX','1996-04-12|155190|21168.23|0.04', '|', null, null, null, null);
<table>
<thead>
<tr>
<th>ENCODED_REGION_NAME</th>
<th>START_KEY</th>
<th>END_KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ff8f9e54519a31e15f264ba6d2b828a4</td>
<td>\xEC\xC1\x15\xAD\xCD\x80\x00\xE1\x06\xEE\x00\xE4V&amp;</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 row selected

splice> CALL SYSCS_UTIL.COMPACT_REGION('SPLICE', 'testTable', 'testIndex', 'ff8f9e54519a31e15f264ba6d2b828a4');
Statement executed.
See Also

» SYCS_UTIL.GET_ENCODED_REGION_NAME
» SYCS_UTIL.GET_REGIONS
» SYCS_UTIL.GET_START_KEY
» SYCS_UTIL.MAJOR_COMPACT_REGION
» SYCS_UTIL.MERGE_REGIONS
SYSCS_UTIL.COMPUTE_SPLIT_KEY

You can 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 pre-split the data that you're importing into HFiles.

Splice Machine recommends using the SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX system procedure instead of this one unless you're an expert user. The combination of using SYSCS_UTIL.COMPUTE_SPLIT_KEY with SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS is exactly equivalent to using SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX.

For more information about splitting your tables and indexes into HFiles, see the Bulk Importing Flat Files section of our Best Practices Guide.

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 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.
This table includes a brief description of each parameter; additional information is available in the Ingestion Parameter Values topic of our Importing Data tutorial.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaName</td>
<td>The name of the schema of the table into which to import.</td>
<td>SPLICE</td>
</tr>
<tr>
<td>tableName</td>
<td>The name of the table into which to import.</td>
<td>playerTeams</td>
</tr>
<tr>
<td>insertColumnList</td>
<td>The names, in single quotes, of the columns to import. If this is null, all columns are imported.</td>
<td>'ID, TEAM'</td>
</tr>
<tr>
<td>NOTE: The individual column names in the insertColumnList do not need to be double-quoted, even if they contain special characters. However, if you do double-quote any column name, you must double-quote all of the column names.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fileOrDirectoryName</td>
<td>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. 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. See the Configuring an S3 Bucket for Splice Machine Access topic for information about accessing data on S3.</td>
<td>/data/mydata/mytable.csv 's3a://splice-benchmark-data/flat/TPCH/100/region'</td>
</tr>
<tr>
<td>oneLineRecords</td>
<td>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.</td>
<td>true</td>
</tr>
<tr>
<td>charset</td>
<td>The character encoding of the import file. The default value is UTF-8.</td>
<td>null</td>
</tr>
<tr>
<td>columnDelimiter</td>
<td>The character used to separate columns. Specify null if using the comma (,) character as your delimiter.</td>
<td>',' ', \t'</td>
</tr>
<tr>
<td>characterDelimiter</td>
<td>The character used to delimit strings in the imported data.</td>
<td>''' ', '''</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>timestampFormat</td>
<td>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: &quot;<code>yyyy-MM-dd HH:mm:ss</code>&quot;. All of the timestamps in the file you are importing must use the same format.</td>
<td>'<code>yyyy-MM-dd HH:mm:ss.SSSS</code>'</td>
</tr>
<tr>
<td>dateFormat</td>
<td>The format of dates 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: &quot;<code>yyyy-MM-dd</code>&quot;.</td>
<td><code>yyyy-MM-dd</code></td>
</tr>
<tr>
<td>timeFormat</td>
<td>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: &quot;<code>HH:mm:ss</code>&quot;.</td>
<td><code>HH:mm:ss</code></td>
</tr>
<tr>
<td>badRecordsAllowed</td>
<td>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.</td>
<td>25</td>
</tr>
<tr>
<td>badRecordDirectory</td>
<td>The directory in which bad record information is logged. Splice Machine logs information to the <code>&lt;import_file_name&gt;.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>. On a cluster, this directory <strong>MUST be on S3, HDFS (or MapR-FS)</strong>. If you're using our Database Service product, files can only be imported from S3.</td>
<td><code>'importErrsDir'</code></td>
</tr>
</tbody>
</table>
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bulkImportDirectory</code> (outputDirectory)</td>
<td>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. For the <code>SYSCS_UTIL.COMPUTE_SPLIT_KEY</code> procedure, where it is named outputDirectory, this parameter specifies the directory into which the split keys are written.</td>
<td><code>hdfs:///tmp/test_hfile_import/</code></td>
</tr>
<tr>
<td><code>skipSampling</code></td>
<td>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. This parameter is only used with the <code>SYSCS_UTIL.BULK_IMPORT_HFILE</code> system procedure.</td>
<td><code>false</code></td>
</tr>
</tbody>
</table>

---

**Usage**

This procedure generates a split keys file in CSV format, which you pass into the `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS`. That procedure pre-splits the data that you then import with `SYSCS_UTIL.BULK_IMPORT_HFILE`.

The functionality of the `SYSCS_UTIL.COMPUTE_SPLIT_KEY` and `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS` procedures has been combined into one simplified procedure: `SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX`. We recommend using the simplified procedure, which performs exactly the same functions.

The Best Practices: Bulk Importing Flat Files section of our *Importing Data Tutorial* describes the different methods for using our bulk HFile import functionality.

**See Also**

- Best Practices: Ingestion
- `SYSCS_UTIL.IMPORT_DATA`
- `SYSCS_UTIL.MERGE_DATA_FROM_FILE`
» SYSCS_UTIL.BULK_IMPORT_HFILE

» SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX

» SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX_AT_POINTS
SYCS_UTIL.SYCS_CREATE_USER

The SYCS_UTIL.SYCS_CREATE_USER system procedure adds a new user account to a database.

This procedure creates users for use with NATIVE authentication.

Syntax

```sql
SYCS_UTIL.SYCS_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.

**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:
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:

```java
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:

```java
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**

- SYCS_UTIL.SYSCS_DROP_USER built-in system procedure
**SYSCS_UTIL.SYSCS_DELETE_BACKUP**

The `SYSCS_UTIL.SYSCS_DELETE_BACKUP` system procedure deletes a previously created backup.

**Syntax**

```sql
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, you can query the `SYS.SYSBACKUP` system table, as described in 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                                    |
----------------------------------------------------------------------------------------------------------------------------------------
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_DELETE_OLD_BACKUPS
- SYSCS_UTIL.SYSCS_RESTORE_DATABASE
- SYSBACKUP
- SYSBACKUPITEMS
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

SYCS_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

» SYCS_UTIL.SYSCS_BACKUP_DATABASE

» SYCS_UTIL.SYSCS_DELETE_BACKUP
» SYSCS_UTIL.SYSCS_RESTORE_DATABASE
» SYSBACKUP
» SYSBACKUPITEMS
SYCS_UTIL.DELETE_REGION

The SYCS_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

SYCS_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 SYCS_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 SYCS_UTIL.DELETE_REGION(),:

- Check region boundaries of the base table and indexes using the SYCS_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 SYCS_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,'\x83');
Statement executed.
splice> call syscs_util.syscs_split_table_or_index_at_points('SPLICE','T','TI','\x83');
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,null);  
<table>
<thead>
<tr>
<th>ENCODED_REGION_NAME</th>
<th>SPLICE_START_KEY</th>
<th>SPLICE_END_KEY</th>
<th>HBASE_START_KEY</th>
<th>HBASE_END_KEY</th>
<th>NUM_HFILES</th>
<th>SIZE</th>
<th>LAST_MODIFICATION_TIME</th>
<th>REGION_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>02953478d84fcba7bb44f3eba0c9036</td>
<td>{ NULL }</td>
<td>{ 3 }</td>
<td></td>
<td></td>
<td>1</td>
<td>1073</td>
<td>2017-12-12 11:12:34.0</td>
<td>splice:1809,,151310595332</td>
</tr>
<tr>
<td>1c1ee3dd90817576ef1148d91666defa</td>
<td>{ 3 }</td>
<td>{ NULL }</td>
<td>1</td>
<td>1073</td>
<td>2017-12-12 11:12:34.0</td>
<td>splice:1809,,151310595332</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 rows selected

splice> call syscs_util.get_regions('SPLICE', 'T', null,null,null,null,null,null,null);  
<table>
<thead>
<tr>
<th>ENCODED_REGION_NAME</th>
<th>SPLICE_START_KEY</th>
<th>SPLICE_END_KEY</th>
<th>HBASE_START_KEY</th>
<th>HBASE_END_KEY</th>
<th>NUM_HFILES</th>
<th>SIZE</th>
<th>LAST_MODIFICATION_TIME</th>
<th>REGION_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>19c21ae5b0b2767403a8beff3148b646</td>
<td>{ NULL, NULL}</td>
<td>{ 3, NULL }</td>
<td>1</td>
<td>1045</td>
<td>2017-12-12 11:12:08.0</td>
<td>splice:1792,,151310592782</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6c8ac07d50cc2e606562dc1949705374</td>
<td>{ 3, NULL }</td>
<td>{ NULL, NULL }</td>
<td>1</td>
<td>1045</td>
<td>2017-12-12 11:12:08.0</td>
<td>splice:1792,,151310592782</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 rows selected

Delete one region from base table and one region from index

splice> call syscs_util.delete_region('SPLICE','T',null,'19c21ae5b0b2767403a8beff3148b646', true);  
Statement executed.

splice> call syscs_util.delete_region('SPLICE','T','TI','02953478d84fcba7bb44f3eba0c9036', true);  
Statement executed.
Verify the results

splice> call syscs_util.get_regions('SPLICE', 'T', 'TI',null,null,null,null,null,null,null,null);  
<table>
<thead>
<tr>
<th>ENCODED_REGION_NAME</th>
<th>SPLICE_START_KEY</th>
<th>SPLICE_END_KEY</th>
<th>HBASE_START_KEY</th>
<th>HBASE_END_KEY</th>
<th>NUM_HFILES</th>
<th>SIZE</th>
<th>LAST_MODIFICATION_TIME</th>
<th>REGION_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>5e8d1ffdf5e8aaa4e85a851caf17a2d9</td>
<td>{ NULL }</td>
<td>{ NULL }</td>
<td></td>
<td></td>
<td>1</td>
<td>1073</td>
<td>2017-12-12 11:14:15.0</td>
<td>splice:1809,,151310605454</td>
</tr>
</tbody>
</table>

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.
```
The SYSCS_UTIL.DISABLE_COLUMN_STATISTICS system procedure disables collection of statistics on a specific table column in your database.

### Syntax

```sql
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('SPLICE', 'Salaries', 'Salary');
Statement executed.
```
See Also

» Data Assignments and Comparisons
» `SYCS_UTIL.ENABLE_COLUMN_STATISTICS`
» `SYCS_UTIL.COLLECT_SCHEMA_STATISTICS`
» `SYCS_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**

```sql
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**

```sql
splice> CALL SYSCS_UTIL.DROP_SCHEMA_STATISTICS('MYSCHEMA');
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
The **SYCS_UTIL.SYCS_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

```
SYCS_UTIL.SYCS_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:

```java
CallableStatement cs = conn.prepareCall
("CALL SYCS_UTIL.SYCS_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

» SYCS_UTIL.SYCS_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**

```java
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

» `SYCS_UTIL.SYCS_EMPTY_STATEMENT_CACHE`

» `SYCS_UTIL.SYCS_INVALIDATE_STORED_STATEMENTS`
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

```sql
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

```java
CallableStatement cs = conn.prepareCall
        ("CALL SYSCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE()");
    cs.execute();
    cs.close();
```

### SQL Example

```sql
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**

```sql
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.
```
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**

SYCS_UTIL.SYCS_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 SYCS_UTIL.SYCS_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
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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOSTNAME</td>
<td>The host on which the server is running.</td>
</tr>
<tr>
<td>PORT</td>
<td>The port on which the server is listening for requests.</td>
</tr>
<tr>
<td>STARTCODE</td>
<td>The system identifier for the Region Server.</td>
</tr>
</tbody>
</table>

### 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

```syntax
SYSCS_UTIL.SYSCS_GET_ALL_PROPERTIES()
```

Results

The displayed results of calling SYSCS_UTIL.SYSCS_GET_ALL_PROPERTIES include these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY</td>
<td>The property name</td>
</tr>
<tr>
<td>VALUE</td>
<td>The property value</td>
</tr>
<tr>
<td>TYPE</td>
<td>The property type</td>
</tr>
</tbody>
</table>
Example
splice> CALL SYSCS_UTIL.SYSCS_GET_ALL_PROPERTIES();

<table>
<thead>
<tr>
<th>KEY</th>
<th>VALU</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>TYPE</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>derivative.authentication.builtin.algorithm</td>
<td>SHA-51</td>
</tr>
<tr>
<td>2</td>
<td>JVM</td>
</tr>
<tr>
<td>derivative.authentication.native.create.credentials.data</td>
<td>true</td>
</tr>
<tr>
<td>J</td>
<td>JVM</td>
</tr>
<tr>
<td>derivative.authentication.provider</td>
<td>NATIVE:spliceDB:LOCAL</td>
</tr>
<tr>
<td>L</td>
<td>JVM</td>
</tr>
<tr>
<td>derivative.connection.requireAuthentication</td>
<td>true</td>
</tr>
<tr>
<td>E</td>
<td>JVM</td>
</tr>
<tr>
<td>derivative.database.collation</td>
<td>UCS_BASIC</td>
</tr>
<tr>
<td>C</td>
<td>DATABASE</td>
</tr>
<tr>
<td>derivative.database.defaultConnectionMode</td>
<td>fullAccess</td>
</tr>
<tr>
<td>S</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derivative.database.propertiesOnly</td>
<td>false</td>
</tr>
<tr>
<td>E</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derivative.database.sqlAuthorization</td>
<td>true</td>
</tr>
<tr>
<td>E</td>
<td>JVM</td>
</tr>
<tr>
<td>derivative.engineType</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derivative.language.logQueryPlan</td>
<td>false</td>
</tr>
<tr>
<td>E</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derivative.language.logStatementText</td>
<td>false</td>
</tr>
<tr>
<td>E</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derivative.language.updateSystemProcs</td>
<td>false</td>
</tr>
<tr>
<td>E</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derivative.locks.escalationThreshold</td>
<td>50</td>
</tr>
<tr>
<td>0</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derivative.storage.propertiesId</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derivative.storage.rowLocking</td>
<td>false</td>
</tr>
<tr>
<td>E</td>
<td>SERVICE</td>
</tr>
<tr>
<td>derivative.stream.error.file</td>
<td>./splice-derby.log</td>
</tr>
<tr>
<td>g</td>
<td>JVM</td>
</tr>
<tr>
<td>splice.authentication</td>
<td>NATIVE</td>
</tr>
<tr>
<td>E</td>
<td>JVM</td>
</tr>
<tr>
<td>splice.debug.logStatementContext</td>
<td>false</td>
</tr>
<tr>
<td>E</td>
<td>JVM</td>
</tr>
<tr>
<td>splice.software.buildtime</td>
<td>2015-10-21 13:18:05</td>
</tr>
<tr>
<td>0</td>
<td>JVM</td>
</tr>
<tr>
<td>splice.software.release</td>
<td>1.5.1</td>
</tr>
<tr>
<td>m</td>
<td>JVM</td>
</tr>
<tr>
<td>splice.software.versionhash</td>
<td>fe1b10bda</td>
</tr>
<tr>
<td>0</td>
<td>JVM</td>
</tr>
</tbody>
</table>

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**

```sql
SYSCS_UTIL.SYSCS_GET_CURRENT_TRANSACTION()
```

**Results**

The displayed results of calling SYSCS_UTIL.SYSCS_GET_CURRENT_TRANSACTION include these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>txnId</td>
<td>The ID of the current transaction</td>
</tr>
</tbody>
</table>

**Example**

```sql
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

```sql
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:

```sql
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
The `SYCS_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 `SYCS_UTIL.MAJOR_COMPACT_REGION`, `SYCS_UTIL.COMPACT_REGION`, or `SYCS_UTIL.MERGE_REGIONS` procedures.

**Syntax**

```sql
SYCS_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 SYCS_UTIL.IMPORT_DATA topic for more information about timestamps.

**dateFormat**

The format of dates 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:

- SYCS_UTIL.COMPACT_REGION
- SYCS_UTIL.MAJOR_COMPACT_REGION
- SYCS_UTIL.MERGE_REGIONS

### Results

The displayed results of calling SYCS_UTIL.SYCS_GET_ENCODED_REGION_NAME include these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCODED_REGION_NAME</td>
<td>The HBase-encoded name of the region.</td>
</tr>
<tr>
<td>START_KEY</td>
<td>The HBase starting key for the region.</td>
</tr>
</tbody>
</table>
**Value** | **Description**
--- | ---
END_KEY | 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);;

<table>
<thead>
<tr>
<th>ENCODED_REGION_NAME</th>
<th>START_KEY</th>
<th>END_KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>8ffc80e3f8ac3b180441371319ea90e2</td>
<td>\x81\x00\x82</td>
<td>\x81\x00\x84</td>
</tr>
</tbody>
</table>

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|0.04', '|', null, null, null, null);

<table>
<thead>
<tr>
<th>ENCODED_REGION_NAME</th>
<th>START_KEY</th>
<th>END_KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ff8f9e54519a31e15f264ba6d2b828a4</td>
<td>\xEC\xC1\x15\xCD\x80\x00\xE1\x06\xEE\x00\xE4</td>
<td>V&amp;</td>
</tr>
</tbody>
</table>

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
SYCS_UTIL.SYCS_GET_LOGGERS

The SYCS_UTIL.SYCS_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**

```sql
SYCS_UTIL.SYCS_GET_LOGGERS()
```

**Results**

The displayed results of calling `SYCS_UTIL.SYCS_GET_LOGGERS` include these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGGERNAME</td>
<td>The name of the logger</td>
</tr>
</tbody>
</table>

**Example**

Here's the output from a call to `SYCS_UTIL.SYCS_GET_LOGGERS`, as of Splice Machine Release 1.5:
splice> CALL SYSCS_UTIL.SYSCS_GET_LOGGERS();
SPICELOGGER

-----------------------------------------------------------------
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.spark.SpliceSpark
com.splicemachine.derby.impl.sql.catalog
com.splicemachine.derby.impl.sql.catalog.SpliceDataDictionary
com.splicemachine.derby.impl.sql.catalog.SpliceDataDictionary.upgrade
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.hbase.HBaseRegionLoads
com.splicemachine.hbase.NoRetryCoprocessorRpcChannel
com.splicemachine.hbase.backup.Backup
com.splicemachine.hbase.backup.BackupBackup
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.SIObserver
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**

```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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGLEVEL</td>
<td>The level of the logger. This is one of the following values, which are described in the <a href="#">Logging</a> topic:</td>
</tr>
</tbody>
</table>

  - 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

SYCS_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. Specify NULL to indicate all regions.

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. Specify NULL to indicate all regions.

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 SYCS_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 SYCS_UTIL.SYCS_GET_REGIONS include these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCODED_REGION_NAME</td>
<td>The HBase-encoded name of the region.</td>
</tr>
<tr>
<td>SPLICE_START_KEY</td>
<td>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 startKey parameter. For example: {1,2}.</td>
</tr>
</tbody>
</table>
### Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPLICE_END_KEY</td>
<td>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 endKey parameter. For example: {1,6}.</td>
</tr>
<tr>
<td>HBASE_START_KEY</td>
<td>The start key for the region, formatted as shown in the HBase Web UI. For example: \x81\x00\x82.</td>
</tr>
<tr>
<td>HBASE_END_KEY</td>
<td>The end key for the region, formatted as shown in the HBase Web UI. For example: \x81\x00\x86.</td>
</tr>
<tr>
<td>NUM_HFILES</td>
<td>The number of HBase Files contained in the region.</td>
</tr>
<tr>
<td>SIZE</td>
<td>The size, in bytes, of the region.</td>
</tr>
<tr>
<td>LAST_MODIFICATION_TIME</td>
<td>The most recent time at which the region was modified.</td>
</tr>
<tr>
<td>REGION_NAME</td>
<td>The unencoded name of the region.</td>
</tr>
</tbody>
</table>

### 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( 'SPLICE','TestTable', null, '1|2', '1|8', '|',null,null,null,null);

<table>
<thead>
<tr>
<th>ENCODED_REGION_NAME</th>
<th>SPLICE_START_KEY</th>
<th>SPLICE_END_KEY</th>
<th>HBASE_START_KEY</th>
<th>HBASE_END_KEY</th>
<th>NUM_HFILES</th>
<th>SIZE</th>
<th>LAST_MODIFICATION_TIME</th>
<th>REGION_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>132c824b9e269006a8e0a3fad577bd12</td>
<td>{ 1, 2}</td>
<td>{ 1, 6}</td>
<td>\x81\x00\x82</td>
<td>\x81\x00\x82</td>
<td>1</td>
<td>1645</td>
<td>2017-08-17 12:44:15.0</td>
<td>splice:2944,\x81\x00\x8</td>
</tr>
<tr>
<td>2,1502999053574.132c824b9e269006a8e0a3fad577bd12.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2ee995a552cbb75b7172eeed27b917cab</td>
<td>{ 1, 6 }</td>
<td>{ 1, 8}</td>
<td>\x81\x00\x86</td>
<td>\x81\x00\x86</td>
<td>1</td>
<td>1192</td>
<td>2017-08-17 08:37:56.0</td>
<td>splice:2944,\x81\x00\x8</td>
</tr>
<tr>
<td>6,1502984266749.2ee995a552cbb75b7172eeed27b917cab.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2 rows selected

To list information about all regions instead, use NULL for the startKey and endKey values:

splice> CALL SYSCS_UTIL.GET_REGIONS( 'SPLICE','TestTable', null, null, null, null, null/null/nul/nul/null/null);
See Also

- `SYCS_UTIL.COMPACT_REGION`
- `SYCS_UTIL.GET_ENCODED_REGION_NAME`
- `SYCS_UTIL.GET_START_KEY`
- `SYCS_UTIL.MAJOR_COMPACT_REGION`
- `SYCS_UTIL.MERGE_REGIONS`
The `SYSCS_UTIL.SYSCS_GET_REGION_SERVER_STATS_INFO` system procedure displays input and output statistics about the cluster.

**Syntax**

```
SYSCS_UTIL.SYSCS_GET_REGION_SERVER_STATS_INFO()
```

**Results**

The displayed results of calling `SYSCS_UTIL.SYSCS_GET_REGION_SERVER_STATS_INFO` include these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST</td>
<td>The host name (or IP address).</td>
</tr>
<tr>
<td>REGIONCOUNT</td>
<td>The number of regions.</td>
</tr>
<tr>
<td>STOREFILECOUNT</td>
<td>The number of files stored.</td>
</tr>
<tr>
<td>WRITEREQUESTCOUNT</td>
<td>The number of write requests.</td>
</tr>
<tr>
<td>READREQUESTCOUNT</td>
<td>The number of read requests.</td>
</tr>
<tr>
<td>TOTALREQUESTCOUNT</td>
<td>The total number of requests.</td>
</tr>
</tbody>
</table>

**Example**

```
splice> CALL SYSCS_UTIL.SYSCS_GET_REGION_SERVER_STATS_INFO();

<table>
<thead>
<tr>
<th>Host</th>
<th>regionCount</th>
<th>storeFileCount</th>
<th>writeRequestCount</th>
<th>readRequestCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>111.222.3.4</td>
<td>58</td>
<td>0</td>
<td>5956</td>
<td>9969</td>
</tr>
<tr>
<td>555.666.7.8</td>
<td>59</td>
<td>0</td>
<td>1723</td>
<td>5702</td>
</tr>
</tbody>
</table>
```

1 row selected
SYCS_UTIL.SYCS_GET_REQUESTS

The SYCS_UTIL.SYCS_GET_REQUESTS system procedure displays information about the number of RPC requests that are coming into Splice Machine.

Syntax

```
SYCS_UTIL.SYCS_GET_REQUESTS()
```

Results

The displayed results of calling SYCS_UTIL.SYCS_GET_REQUESTS include these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOSTNAME</td>
<td>The host name.</td>
</tr>
<tr>
<td>PORT</td>
<td>The port receiving requests.</td>
</tr>
<tr>
<td>TOTALREQUESTS</td>
<td>The total number of RPC requests on that port.</td>
</tr>
</tbody>
</table>

Example

```
splice> CALL SYCS_UTIL.SYCS_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**

```plaintext
SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS();
```

**Security Note**

This procedure runs another, internal (undocumented) system procedure named `SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS_LOCAL`. This means that when you change permissions for `SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS`, you must also make the same permission changes to `SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS_LOCAL`.

*NOTE:* This will be handled automatically in a future release.

**Results**

The displayed results of calling `SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS` include these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UUID</td>
<td>The operation identifier. This is the same identifier that is shown in the Spark console.</td>
</tr>
<tr>
<td>USER</td>
<td>The name of the database user.</td>
</tr>
<tr>
<td>HOSTNAME</td>
<td>The host on which the server is running.</td>
</tr>
<tr>
<td>SESSION</td>
<td>The session ID.</td>
</tr>
<tr>
<td>SQL</td>
<td>The SQL statement that is running.</td>
</tr>
<tr>
<td>SUBMITTED</td>
<td>The date and time that the operation was submitted.</td>
</tr>
<tr>
<td>ELAPSED</td>
<td>Elapsed time since the operation began running.</td>
</tr>
</tbody>
</table>
### Example

splice> call SYSCS_UTIL.SYSCS_GET_RUNNING_OPERATIONS();

<table>
<thead>
<tr>
<th>UUID</th>
<th>USER</th>
<th>HOSTNAME</th>
<th>SESSION</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>34b0f479-be9a-4933-9b4d-900af218a19c</td>
<td>SPLICE</td>
<td>MacBook-Pro.local:1527</td>
<td>264</td>
<td><code>select * from sys.systables -splice- properties useSpark=true</code></td>
</tr>
<tr>
<td>4099f016-3c9d-4c62-8059-ff18d3b38a19</td>
<td>SPLICE</td>
<td>MacBook-Pro.local:1527</td>
<td>4</td>
<td><code>call syscs_util.syscs_get_running_operations</code></td>
</tr>
</tbody>
</table>

2 rows selected

splice> call SYCS_UTIL.SYCS_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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMANAME</td>
<td>The schema to which the table belongs.</td>
</tr>
<tr>
<td>TABLENAME</td>
<td>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.</td>
</tr>
<tr>
<td>ISINDEX</td>
<td>A Boolean value that specifies whether the HBase table is an index table.</td>
</tr>
<tr>
<td>HBASEREGIONS</td>
<td>The HBase regions on which the table resides. There can be multiple regions.</td>
</tr>
<tr>
<td></td>
<td>Each region display shows (tableName, regionId.storeFileSize, memStoreSize, and storeIndexSize MB).</td>
</tr>
</tbody>
</table>
splice> CALL SYSCS_UTIL.SYSCS_GET_SCHEMA_INFO();

<table>
<thead>
<tr>
<th>SCHEMANAME</th>
<th>TABLNAME</th>
<th>REGIONNAME</th>
<th>MEMSTORESIZE</th>
<th>STOREINDEXSIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPLICE</td>
<td>PLAYERS</td>
<td>2176,,1446847689610.7211e284f7f767d7b142dbd639b4d9bf.</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td>SPLICE</td>
<td>PITCHING</td>
<td>1968,,1446260714743.01963d7260fc9d4dc01507eccdf67e40.</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td>SPLICE</td>
<td>BATTING</td>
<td>1984,,1446260731076.ca29785eb5b16a8752d9c4ceeaad2ce4.</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td>SPLICE</td>
<td>FIELDING</td>
<td>2192,,1447092732332.b4aae9902302bbac1a08432e6ff2df.</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td>SPLICE</td>
<td>SALARIES</td>
<td>2256,,1447803176538.11ce38c9e470b4d209de4d32c96cb815.</td>
<td>false</td>
<td>0</td>
</tr>
</tbody>
</table>

5 rows selected
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_INFO()` include these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOSTNAME</td>
<td>The identity of your Splice Machine connection.</td>
</tr>
<tr>
<td>SESSION</td>
<td>The ID of your database connection session.</td>
</tr>
</tbody>
</table>

**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:
<table>
<thead>
<tr>
<th>Job Id (Job Group)</th>
<th>Description</th>
<th>Submitted</th>
<th>Duration</th>
<th>Stages: Succeeded/Total</th>
<th>Tasks (for all stages): Succeeded/Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (SPLICE <a href="">localhost:1527,4,e5ff1a51-4ac1-4202-a74c-a2d54ab3525a,36096</a>)</td>
<td>select * from sys.systables --splice-properties useSpark=true (kill) Produce Result Set</td>
<td>2017/08/25 14:03:07</td>
<td>8s</td>
<td>0/1</td>
<td>0/1</td>
</tr>
</tbody>
</table>
**SYSCS_UTIL.GET_START_KEY**

The `SYSCS_UTIL.GET_START_KEY` system procedure that retrieves the starting key value, in unencoded format, for a specified HBase region.

**Syntax**

```sql
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:
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', 'b35fe82916cdd1d48bb5c43f60a9b8b5');
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
The **SYCS_UTIL.SYCS_GET_VERSION_INFO** system procedure displays the version of Splice Machine installed on each node in your cluster.

**Syntax**

```sql
SYCS_UTIL.SYCS_GET_VERSION_INFO()
```

**Results**

This procedure does not return a result.

**Example**

```sql
splice> call SYCS_UTIL.SYCS_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 **SYCS_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**

```
SYCS_UTIL.SYSCS_GET_WRITE_INTAKE_INFO()
```

**Results**

The displayed results of calling **SYCS_UTIL.SYSCS_GET_WRITE_INTAKE_INFO** include these values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOSTNAME</td>
<td>The host name.</td>
</tr>
<tr>
<td>ACTIVEWRITETHREADS</td>
<td>The number of active write threads.</td>
</tr>
<tr>
<td>COMPACTIONQUEUESIZELIMIT</td>
<td>The compaction queue limit at which writes will be blocked.</td>
</tr>
<tr>
<td>FLUSHQUEUELIMIT</td>
<td>The flush queue limit at which writes will be blocked.</td>
</tr>
<tr>
<td>IPCRESERVEDPOOL</td>
<td>The number of IPC threads reserved for reads.</td>
</tr>
<tr>
<td></td>
<td>The maximum number of bulk writes that are allowed currently is equal to</td>
</tr>
<tr>
<td></td>
<td>the total number of IPC threads minus this value.</td>
</tr>
</tbody>
</table>
Example

splice> CALL SYSCS_UTIL.SYSCS_GET_WRITE_INTAKE_INFO();

<table>
<thead>
<tr>
<th>host</th>
<th>depThreads</th>
<th>indThreads</th>
<th>depCount</th>
<th>indCount</th>
<th>avgThroughput</th>
<th>oneMinAvgThroughput</th>
<th>fiveMinAvgThroughput</th>
<th>fifteenMinAvgThroughput</th>
<th>totalRejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>localhost:55709</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.01145109332258</td>
<td>5.472367185912236E-4</td>
<td>0.007057900046373111</td>
<td>0.0053884511108858845</td>
<td>0</td>
</tr>
</tbody>
</table>

1 row selected
SYCS_UTIL.IMPORT_DATA

The SYCS_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.

This procedure is one of several built-in system procedures provided by Splice Machine for importing data into your database. See our Best Practices: Ingestion for help with selecting the right process for your situation.

Syntax

call SYCS_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

This table includes a brief description of each parameter; additional information is available in the Ingestion Parameter Values topic of our Importing Data tutorial.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaName</td>
<td>The name of the schema of the table into which to import.</td>
<td>SPLICE</td>
</tr>
<tr>
<td>tableName</td>
<td>The name of the table into which to import.</td>
<td>playerTeams</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>insertColumnList</td>
<td>The names, in single quotes, of the columns to import. If this is null, all columns are imported.</td>
<td>'ID, TEAM'</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> The individual column names in the insertColumnList do not need to be double-quoted, even if they contain special characters. However, if you do double-quote any column name, you must double-quote all of the column names.</td>
<td></td>
</tr>
<tr>
<td>fileOrDirectoryName</td>
<td>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. 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. See the Configuring an S3 Bucket for Splice Machine Access topic for information about accessing data on S3.</td>
<td>/data/mydata/mytable.csv 's3a://splice-benchmark-data/flat/TPCH/100/region'</td>
</tr>
<tr>
<td>columnDelimiter</td>
<td>The character used to separate columns. Specify null if using the comma (,) character as your delimiter.</td>
<td>'</td>
</tr>
<tr>
<td>characterDelimiter</td>
<td>The character used to delimit strings in the imported data.</td>
<td>'&quot;', '&quot;'</td>
</tr>
<tr>
<td>timestampFormat</td>
<td>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 Java.sql.Timestamp default format, which is: &quot;yyyy-MM-dd HH:mm:ss&quot;. All of the timestamps in the file you are importing must use the same format.</td>
<td>'yyyy-MM-dd HH:mm:ss.SSZ'</td>
</tr>
<tr>
<td>dateFormat</td>
<td>The format of dates 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: &quot;yyyy-MM-dd&quot;.</td>
<td>yyyy-MM-dd</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeFormat</td>
<td>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: &quot;HH:mm:ss&quot;.</td>
<td>HH:mm:ss</td>
</tr>
<tr>
<td>badRecordsAllowed</td>
<td>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.</td>
<td>25</td>
</tr>
<tr>
<td>badRecordDirectory</td>
<td>The directory in which bad record information is logged. Splice Machine logs information to the <code>&lt;import_file_name&gt;.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>. 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.</td>
<td>'importErrsDir'</td>
</tr>
<tr>
<td>oneLineRecords</td>
<td>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.</td>
<td>true</td>
</tr>
<tr>
<td>charset</td>
<td>The character encoding of the import file. The default value is UTF-8.</td>
<td>null</td>
</tr>
</tbody>
</table>

### Results

SYCS_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

The SYSCS_UTIL.SYSCS_MERGE_DATA_FROM_FILE imports new records into your database in the same way as does the SYSCS_UTIL.IMPORT_DATA procedure. SYSCS_UTIL.SYSCS_MERGE_DATA_FROM_FILE can also update existing records in your database; for this to work, the table you’re importing into must have a primary key. Because this procedure has to determine if a record already exists and how to update it, MERGE_DATA is slightly slower than using IMPORT_DATA; if you know that you’re ingesting all new records, you’ll get better performance with IMPORT_DATA.

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:

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

Examples

This section presents a couple simple examples.

The Importing Flat Files 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:
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:

```sql
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 Flat Files for more examples.

See Also

- Best Practices: Ingestion
- Importing Flat Files
- `SYSCS_UTIL.MERGE_DATA_FROM_FILE`
- `SYSCS_UTIL.BULK_HFILE_IMPORT`
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).

- If you’re using Splice Machine on a cluster, the Jar file can be in the HDFS file system or in an S3a bucket on Amazon AWS.
- If you’re using our standalone version, the Jar file can be in an S3a bucket on Amazon AWS or in your local file system.

Here are a few examples:

- https://s3a.amazonaws.com/splice/examples/jars/tours.jar
- hdfs:///home/jars/tours.jar
- 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', 'SPLICE.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

```sql
-- 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](#)
The SYSCS_UTIL.INVALIDATE_DICTIONARY_CACHE system procedure invalidates the dictionary cache on the connection’s region server.

Syntax

```sql
SYSCS_UTIL.INVALIDATE_DICTIONARY_CACHE()
```

Results

This procedure does not return a result.

Example

```sql
splice> CALL SYSCS_UTIL.INVALIDATE_DICTIONARY_CACHE();
Statement executed.
```

See Also

- `SYSCS_UTIL.INVALIDATE_GLOBAL_DICTIONARY_CACHE`
SYCS_UTIL.INVALIDATE_GLOBAL_DICTIONARY_CACHE

The SYCS_UTIL.INVALIDATE_GLOBAL_DICTIONARY_CACHE system procedure invalidates the dictionary cache on all region servers.

Syntax

SYCS_UTIL.INVALIDATE_GLOBAL_DICTIONARY_CACHE()

Results

This procedure does not return a result.

Example

splice> CALL SYCS_UTIL.INVALIDATE_GLOBAL_DICTIONARY_CACHE();
Statement executed.

See Also

SYCS_UTIL.INVALIDATE_DICTIONARY_CACHE
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**

```sql
SYCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS()
```

**Results**

This procedure does not return a result.

**Example**

```sql
splice> CALL SYCS_UTIL.SYSCS_INVALIDATE_STORED_STATEMENTS();
Statement executed.
```

**See Also**

- `SYCS_UTIL.SYCS_EMPTY_GLOBAL_STATEMENT_CACHE`
- `SYCS_UTIL.SYCS_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();
+-----------------+---------+-------------------------------+----------+
| UUID            | USER    | HOSTNAME                      | SESSION  |
|-----------------|---------+-------------------------------+----------|
| bf610dea-d33e-4304-bf2e-4f10e667aa98 | SPLICE  | localhost:1528                | 2        |
| 33567e3c-ef33-46dc-8d10-5ceb79348c2e  | SPLICE  | localhost:1528                | 20       |
+-----------------+---------+-------------------------------+----------+
splice> call SYSCS_UTIL.SYSCS_KILL_OPERATION('33567e3c-ef33-46dc-8d10-5ceb79348c2e');
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 SYCS_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-compactated.

Syntax

```
SYCS_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 SYCS_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:

```sql
splice> CALL SYSCS_UTIL.GET_ENCODED_REGION_NAME('SPLICE', 'TESTTABLE',
null, '1|2', '|', null, null, null, null);
```

<table>
<thead>
<tr>
<th>ENCODED_REGION_NAME</th>
<th>START_KEY</th>
<th>END_KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>8ffc80e3f8ac3b180441371319ea90e2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 row selected

```sql
splice> CALL SYSCS_UTIL.COMPACT_REGION('SPLICE', '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:

```sql
splice> CALL SYSCS_UTIL.GET_ENCODED_REGION_NAME('SPLICE', 'TESTTABLE',
'SHIP_INDEX', '1996-04-12|155190|21168.23|0.04',
'|', null, null, null, null);
```

<table>
<thead>
<tr>
<th>ENCODED_REGION_NAME</th>
<th>START_KEY</th>
<th>END_KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ff8f9e54519a31e15f264ba6d2b828a4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 row selected

```sql
splice> CALL SYSCS_UTIL.COMPACT_REGION('SPLICE', 'testTable',
'testIndex', 'ff8f9e54519a31e15f264ba6d2b828a4');
```

Statement executed.
See Also

» SYSCS_UTIL.COMPACT_REGION
» SYCS_UTIL.GET_ENCODED_REGION_NAME
» SYCS_UTIL.GET_REGIONS
» SYCS_UTIL.GET_START_KEY
» SYCS_UTIL.MERGE_REGIONS
**SYCS_UTIL.MERGE_DATA_FROM_FILE**

The SYCS_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.

This procedure is one of several built-in system procedures provided by Splice Machine for importing data into your database. See our Best Practices: Ingestion for help with selecting the right process for your situation.

**Syntax**

```sql
call SYCS_UTIL.MERGE_DATA_FROM_FILE (  
   schemaName,  
   tableName,  
   insertColumnList | null,  
   fileName,  
   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 `SYCS_UTIL.MERGE_DATA_FROM_FILE`. Each parameter name links to a more detailed description in our Ingestion Parameter Values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaName</td>
<td>The name of the schema of the table into which to import.</td>
<td>SPLICE</td>
</tr>
<tr>
<td>tableName</td>
<td>The name of the table into which to import.</td>
<td>playerTeams</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>insertColumnList</td>
<td>The names, in single quotes, of the columns to import. If this is null, all</td>
<td>'ID, TEAM'</td>
</tr>
<tr>
<td></td>
<td>columns are imported.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> The individual column names in the insertColumnList do not need</td>
<td></td>
</tr>
<tr>
<td></td>
<td>to be double-quoted, even if they contain special characters. However, if</td>
<td></td>
</tr>
<tr>
<td></td>
<td>you do double-quote any column name, <strong>you must</strong> double-quote all of the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>column names.</td>
<td></td>
</tr>
<tr>
<td>fileName</td>
<td>The file is imported; you can import compressed or uncompressed files.</td>
<td>/data/mydata/mytable.csv</td>
</tr>
<tr>
<td></td>
<td>On a cluster, the file to be imported MUST be on S3, HDFS (or MapR-FS). If</td>
<td></td>
</tr>
<tr>
<td></td>
<td>you're using our Database Service product, the file can only be imported</td>
<td></td>
</tr>
<tr>
<td></td>
<td>from S3.</td>
<td></td>
</tr>
<tr>
<td>columnDelimiter</td>
<td>The character used to separate columns, Specify null if using the comma (,)</td>
<td>'</td>
</tr>
<tr>
<td></td>
<td>character as your delimiter.</td>
<td></td>
</tr>
<tr>
<td>characterDelimiter</td>
<td>The character used to delimit strings in the imported data.</td>
<td>'' ', '''</td>
</tr>
<tr>
<td>timestampFormat</td>
<td>The format of timestamps stored in the file. You can set this to null if</td>
<td>'yyyy-MM-dd HH:mm:ss.SSZ'</td>
</tr>
<tr>
<td></td>
<td>there are no time columns in the file, or if the format of any timestamps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in the file match the Java.sql.Timestamp default format, which is: &quot;yyyy-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MM-dd HH:mm:ss.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>All of the timestamps in the file you are importing must use the same</td>
<td></td>
</tr>
<tr>
<td></td>
<td>format.</td>
<td></td>
</tr>
<tr>
<td>dateFormat</td>
<td>The format of dates stored in the file. You can set this to null if there</td>
<td>yyyy-MM-dd</td>
</tr>
<tr>
<td></td>
<td>are no date columns in the file, or if the format of any dates in the file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>match pattern: &quot;yyyy-MM-dd&quot;.</td>
<td></td>
</tr>
<tr>
<td>timeFormat</td>
<td>The format of time values stored in the file. You can set this to null if</td>
<td>HH:mm:ss</td>
</tr>
<tr>
<td></td>
<td>there are no time columns in the file, or if the format of any times in the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>file match pattern: &quot;HH:mm:ss&quot;.</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>badRecordsAllowed</td>
<td>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.</td>
<td>25</td>
</tr>
<tr>
<td>badRecordDirectory</td>
<td>The directory in which bad record information is logged. Splice Machine logs information to the <code>&lt;import_file_name&gt;</code>.bad 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/</code>foo.csv.bad. On a cluster, this directory <strong>MUST be on S3, HDFS (or MapR-FS)</strong>. If you're using our Database Service product, files can only be imported from S3.</td>
<td>'importErrsDir'</td>
</tr>
<tr>
<td>oneLineRecords</td>
<td>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.</td>
<td>true</td>
</tr>
<tr>
<td>charset</td>
<td>The character encoding of the import file. The default value is UTF-8.</td>
<td>null</td>
</tr>
</tbody>
</table>

**NOTE:** The `SYSCS_UTIL.MERGE_DATA_FROM_FILE` procedure only imports single files; it does not process directories.

**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**

The `SYSCS_UTIL.SYSCS_MERGE_DATA_FROM_FILE` imports new records into your database in the same way as does the `SYSCS_UTIL.IMPORT_DATA` procedure. `SYSCS_UTIL.SYSCS_MERGE_DATA_FROM_FILE` can also update existing records in your database; for this to work, the table you’re importing into must have a primary key. Because this procedure has to determine if a record already exists and how to update it, `MERGE_DATA` is slightly slower than using `IMPORT_DATA`; if you know that you’re ingesting all new records, you’ll get better performance with `IMPORT_DATA`.

**Record Import Failure Reasons**

When merging 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 Flat Files 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:
Example 2: Basic Merge of a Flat File

Here's a very basic example of using `MERGE_DATA_FROM_FILE` to add new records and update a few existing records in a table. This example ingests into the same table that we just used in the `IMPORT_DATA` example above.

1. **Access a simple file named `mergetest.csv` from an S3 bucket on AWS.** That file contains the following data. Note that the rows with key values 2 and 4 already exist in the table:

   2 22
   4 44
   5 55
   6 66

2. **Use `MERGE_DATA` to import that data into the `testImport` table:**

   ```sql
   splice> CALL SYSCS_UTIL.MERGE_DATA_FROM_FILE('TEST', 'testImport', null, 's3a:/mypublicbucket/mergetest.csv', ',', null, null, null, null, 0, 'hdfs:///tmp/test_import/', false, null);
   ```

<table>
<thead>
<tr>
<th>rowsUpdated</th>
<th>rowsInserted</th>
<th>failedRows</th>
<th>files</th>
<th>dataSize</th>
<th>failedLog</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>20</td>
<td>NONE</td>
</tr>
</tbody>
</table>

3. **Use a `SELECT` statement to verify that all went well:**
Note that this MERGE_DATA_FROM_FILE call uses exactly the same parameter values as does the previous call to IMPORT_DATA, with the exception of importing a different file. As you can see, two rows (A1=2 and A1=4) were updated with new B1 values, and two new rows were added by this merge call.

**Example 3: 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('SPLICE','MYTABLE',null,'data.csv','\t','''',null,null,null,0,'BAD', false, null);
```

See Importing Flat Files for more examples.

**See Also**

- Best Practices: Ingestion
- Importing Flat Files
- `SYSCS_UTIL.IMPORT_DATA`
The `SYCS_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 `SYCS_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

```sql
SYCS_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 `SYCS_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 `SYCS_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', '5a59b4a46a8a07180a469dbe0b40fad', '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

```java
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!: 
See Also

`SYCS_UTIL.SYCS_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

```sql
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
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.

A major compaction actually reads every block of data from the every store file in a Region, and rewrites only the live data to a single store file. This permanently deletes the rows that were previously marked as deleted. HBase runs major compactions on a scheduled interval, which is specified in the hbase.hregion.majorcompaction property; the default value for this property in Splice Machine is 7 days.

Splice Machine recommends running a major compaction on a schema after you've imported an entire database.

For more information about compactions, see Using Compaction and Vacuuming.

**Syntax**

```
SYCS_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 SYCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA('SPlice');
Statement executed.
```

**See Also**

- SYCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE
**SYCS_UTIL.SYCS_PERFORM_MAJOR_COMPACTION_ON_TABLE**

The `SYSCS_UTIL.SYCS_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.

A *major compaction* actually reads every block of data from the every store file in a Region, and rewrites only the live data to a single store file. This permanently deletes the rows that were previously marked as deleted. HBase runs major compactions on a scheduled interval, which is specified in the `hbase.hregion.majorcompaction` property; the default value for this property in Splice Machine is 7 days.

Splice Machine recommends running a major compaction on a table after you've imported a large dataset into the table or deleted a large number of rows from the table in your database.

For more information about compactions, see [Using Compaction and Vacuuming](#).

**Syntax**

```sql
SYCS_UTIL.SYCS_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**

```sql
splice> CALL SYCS_UTIL.SYCS_PERFORM_MAJOR_COMPACTION_ON_TABLE('SPI', 'Pitching');
Statement executed.
```
See Also

» SYSCS_UTIL.SYSCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA
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**

```java
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`:

```sql
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**

```sql
SQLJ.REMOVE_JAR(
    qualified_jar_name VARCHAR(32672),
    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
-- 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:

```
MYSCHEMA.Sample1
    -- a delimited identifier.
MYSCHEMA."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
-- 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](#)
SYCS_UTIL.SYCS_RESET_PASSWORD

The SYCS_UTIL.SYCS_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

```
SYCS_UTIL.SYCS_RESET_PASSWORD(IN userName VARCHAR(128),
                               IN password VARCHAR(32672))
```

**user Name**

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 SYCS_UTIL.SYCS_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 SYCS_UTIL.SYSCS_RESET_PASSWORD('fred', 'temppassword');
Statement executed.

Reset the password of a user named MrBaseball:

splice> CALL SYCS_UTIL.SYSCS_RESET_PASSWORD('MrBaseball', 'baseball!');
Statement executed.

**See Also**

» SYCS_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.

**ENTERPRISE ONLY:** This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product; contact Splice Machine Sales for information.

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.
- The restore runs asynchronously, which means that you **need to** look at the region server log for a message that the restore is complete, and then reboot 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**

```sql
SYCS_UTIL.SYSCS_RESTORE_DATABASE( VARCHAR backupDir,
                                BIGINT  backupId,
                                BOOLEAN validate );
```

**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. To find the `backupId`, you can query the `SYS.SYSBACKUP` system table, as described in the Backing Up and Restoring topic.

**validate**

A Boolean value that specifies whether to validate the backup before restoring from it:

- If `validate` is `false`, the restore proceeds without any pre-validation.
- If `validate` is `true`, the backup is validated before the restoration is started. (See `SYSCS_UTIL.VALIDATE_BACKUP`). If the validation check finds inconsistencies, the errors are reported to the user, and the database is not restored. If the inconsistencies are minor, you can choose to re-run this procedure with `validate` set to `false`.

## Backup and Restore Compatibility

Note that you can only use specific restore procedures with specific types of backups. For example, you can use the `RESTORE_TABLE` procedure to restore from a backup created by `BACKUP_TABLE`, but you cannot use `RESTORE_TABLE` to restore from a backup created by `BACKUP_DATABASE`. The following table summarizes backup-restore compatibility:

<table>
<thead>
<tr>
<th>If you backed up with this procedure:</th>
<th>You can restore with these procedures:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYCS_UTIL.SYCS_UTIL.SYSCS_BACKUP_DATABASE</td>
<td>SYCS_UTIL.SYSCS_UTIL.SYSCS_RESTORE_DATABASE</td>
</tr>
<tr>
<td>SYCS_UTIL.SYCS_UTIL.SYSCS_BACKUP_SCHEMA</td>
<td>SYCS_UTIL.SYSCS_UTIL.SYSCS_RESTORE_SCHEMA, SYCS_UTIL.SYSCS_UTIL.SYSCS_RESTORE_TABLE</td>
</tr>
<tr>
<td>SYCS_UTIL.SYSCS_UTIL.SYSCS_BACKUP_TABLE</td>
<td>SYCS_UTIL.SYSCS_UTIL.SYSCS_RESTORE_TABLE</td>
</tr>
</tbody>
</table>

## 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.

You must look at the region server log for a message that the restore is complete, and then restart 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**
This section includes two examples, both of which perform a pre-restore validation of the backup.

Stop using your database before restoring, and keep in mind that restoring a database may take several minutes, depending on the size of your database.

**Example 1: Successful Restoration**
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.
splice> SELECT * FROM SYS.SYSBACKUP;

<table>
<thead>
<tr>
<th>BACKUP_ID</th>
<th>BEGIN_TIMESTAMP</th>
<th>END_TIMESTAMP</th>
<th>STATUS</th>
<th>FILESYSTEM</th>
</tr>
</thead>
</table>

2 rows selected

splice> CALL SYSCS_UTIL.SYSCS_RESTORE_DATABASE('./dbBackups/', 74101, true);
Statement executed.

Once the restoration is complete, reboot your database by the Starting Your Database.

**Example 2: Validation Failure**
Here's a similar restore attempt that terminates after finding inconsistencies in the backup during validation:

splice> SELECT * FROM SYS.SYSBACKUP;

<table>
<thead>
<tr>
<th>BACKUP_ID</th>
<th>BEGIN_TIMESTAMP</th>
<th>END_TIMESTAMP</th>
<th>STATUS</th>
<th>FILESYSTEM</th>
</tr>
</thead>
</table>

2 rows selected

splice> CALL SYSCS_UTIL.SYSCS_RESTORE_DATABASE('./dbBackups/', 63541, true);
result | warnings
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BR010</td>
<td>A data file ./dbBackups/BACKUP_63541/tables/SPLICE_TXN/f4460c47f6c96fe8d76c0def11c22dc8/V/c7350delacaf4a11a561472675edaldd is missing. The restored table may be corrupted.</td>
</tr>
</tbody>
</table>

Found inconsistencies in backup | To force a restore, set validate to false

2 rows selected

**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.VALIDATE_BACKUP
» SYSBACKUP
» SYSBACKUPITEMS
SYSCS_UTIL.SYSCS_RESTORE_DATABASE_TO_TIMESTAMP

The SYSCS_UTIL.SYSCS_RESTORE_DATABASE_TO_TIMESTAMP system procedure restores your database to the state it was in when a backup was performed at the closest time before the specified timestamp.

**Enterprise Only:** This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product; contact Splice Machine Sales for information.

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**

```java
SYCS_UTIL.SYSCS_RESTORE_DATABASE_TO_TIMESTAMP( VARCHAR backupDir,
        BIGINT backupId,
        BOOLEAN validate,
        VARCHAR pointInTime );
```

**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. To find the `backupId`, you can query the `SYS.SYSBACKUP` system table, as described in the Backing Up and Restoring topic.

**validate**

A Boolean value that specifies whether to validate the backup before restoring from it:

- If `validate` is `false`, the restore proceeds without any pre-validation.
- If `validate` is `true`, the backup is validated before the restoration is started. (See `SYSCS_UTIL.VALIDATE_BACKUP`). If the validation check finds inconsistencies, the errors are reported to the user, and the database is *not* restored. If the inconsistencies are minor, you can choose to re-run this procedure with `validate` set to `false`.

**pointInTime**

A VARCHAR or string value that specifies the timestamp, in `yyyy-MM-dd HH:mm:ss` format.

You can find the timestamp of a specific SQL statement in the `splice-derby.log` file.

### Usage

Restoring you 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
This section includes two examples, both of which perform a pre-restore validation of the backup.

Stop using your database before restoring, and keep in mind that restoring a database may take several minutes, depending on the size of your database.

Example 1: Successful Restoration
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 to the specified timestamp (2015-11-30 19:30:00).

splice> SELECT * FROM SYS.SYSBACKUP;
BACKUP_ID | BEGIN_TIMESTAMP          | END_TIMESTAMP            | STATUS    | FILESYSTEM |
M          | SCOPE | INCR | INCREMENTAL_PARENT | BACKUP_ITEM |
--------------------------------------------------------------------------------------------------------
s/          | true  | 40975 |                      | 30          |
e          | false | -1   |                      | 93          |
2 rows selected

splice> CALL SYSCS_UTIL.SYSCS_RESTORE_DATABASE_TO_TIMESTAMP('./dbBackups/', 74101, true, '2015-11-30 19:30:00');
Statement executed.

Once the restoration is complete, reboot your database by the Starting Your Database.

Example 2: Validation Failure
Here’s a similar restore attempt that terminates after finding inconsistencies in the backup during validation:
splice> SELECT * FROM SYS.SYSBACKUP;

<table>
<thead>
<tr>
<th>BACKUP_ID</th>
<th>BEGIN_TIMESTAMP</th>
<th>END_TIMESTAMP</th>
<th>STATUS</th>
<th>FILESYTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>SCOPE</td>
<td>INCR &amp;</td>
<td>INCREMENTAL_PARENT &amp;</td>
<td>BACKUP_ITEM</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>--------</td>
<td>---------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>

2 rows selected


result | warnings

BR010 | A data file ./dbBackups/BACKUP_63541/tables/SPL ICE_TXN/f4460c47f6c96fe8d76c0def11c22dc8/V/c7350delacaf4a11a561472675edaldd is missing. The restored table may be corrupted. Found inconsistencies in backup | To force a restore, set validate to false

2 rows selected

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_VALIDATE_BACKUP
» SYSBACKUP
» SYSBACKUPITEMS
SYSCS_UTIL.SYSCS_RESTORE_DATABASE_TO_TRANSACTION

The SYSCS_UTIL.SYSCS_RESTORE_DATABASE_TO_TRANSACTION system procedure restores your database to the state it was in when a backup was performed after a specified transaction ID completed.

**ENTERPRISE ONLY:** This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product; contact Splice Machine Sales for information.

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**

```sql
SYSCS_UTIL.SYSCS_RESTORE_DATABASE_TO_TRANSACTION(
    VARCHAR backupDir,
    BIGINT backupId,
    BOOLEAN validate,
    BIGINT transactionId );
```

**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. To find the `backupId`, you can query the `SYS.SYSBACKUP` system table, as described in the Backing Up and Restoring topic.

**validate**

A Boolean value that specifies whether to validate the backup before restoring from it:

- If `validate` is `false`, the restore proceeds without any pre-validation.
- If `validate` is `true`, the backup is validated before the restoration is started. (See `SYSCS_UTIL.VALIDATE_BACKUP`). If the validation check finds inconsistencies, the errors are reported to the user, and the database is *not* restored. If the inconsistencies are minor, you can choose to re-run this procedure with `validate` set to `false`.

**transactionID**

The ID of a completed transaction. You can find the transaction ID of a specific SQL statement in the `splice-derby.log` file.

**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

This section includes two examples, both of which perform a pre-restore validation of the backup.

Stop using your database before restoring, and keep in mind that restoring a database may take several minutes, depending on the size of your database.

Example 1: Successful Restoration

Assuming that you've already discovered the ID of the transaction to which you want to restore (765123 in the example), this 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, specifying the transaction ID.

```
splice> SELECT * FROM SYS.SYSBACKUP;
```

```
<table>
<thead>
<tr>
<th>BACKUP_ID</th>
<th>BEGIN_TIMESTAMP</th>
<th>END_TIMESTAMP</th>
<th>STATUS</th>
<th>FILESYSTEM</th>
</tr>
</thead>
</table>
```

```
splice> CALL SYSCS_UTIL.SYSCS_RESTORE_DATABASE_TO_TRANSACTION('./dbBackups/', 74101, true, 765123);
```

Once the restoration is complete, reboot your database by the Starting Your Database.

Example 2: Validation Failure

Here's a similar restore attempt that terminates after finding inconsistencies in the backup during validation:
splice> SELECT * FROM SYS.SYSBACKUP;

```
<table>
<thead>
<tr>
<th>BACKUP_ID</th>
<th>BEGIN_TIMESTAMP</th>
<th>END_TIMESTAMP</th>
<th>STATUS</th>
<th>FILESYSTEM</th>
</tr>
</thead>
</table>
```

2 rows selected

splice> CALL SYSCS_UTIL.SYSCS_RESTORE_DATABASE('./dbBackups/', 63541, true, 443225);

```
<table>
<thead>
<tr>
<th>result</th>
<th>warnings</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR010</td>
<td></td>
</tr>
<tr>
<td>A data file ./dbBackups/BACKUP_63541/tables/SPLICE_TXN/f4460c47f6c96fe8d76c0def11c22dc8/V/c7350delacaf4a11a561472675edaldd is missing. The restored table may be corrupted. Found inconsistencies in backup</td>
<td>To force a restore, set validate to false</td>
</tr>
</tbody>
</table>
```

2 rows selected

**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.VALIDATE_BACKUP**
- **SYSBACKUP**
- **SYSBACKUPITEMS**
The SYSCS_UTIL.SYSCS_RESTORE_SCHEMA system procedure restores a schema that was previously backed up with the SYSCS_UTIL.SYSCS_BACKUP_SCHEMA procedure. You can restore the schema to another schema on the same cluster, or on a different cluster. The schema and all of its tables and other objects are created anew in your database.

**NOTE:** The schema to which you are restoring must not already exist in the database; if it does, RESTORE_SCHEMA will not proceed.

**ENTERPRISE ONLY:** This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product; [contact Splice Machine Sales](mailto:contact@splicemachine.com) for information.

### Syntax

```sql
SYSCS_UTIL.SYSCS_RESTORE_SCHEMA(   VARCHAR destSchema,
   VARCHAR sourceSchema,
   VARCHAR directory,
   BIGINT  backupId,
   BOOLEAN validate );
```

- **destSchema**
  The name of the schema to which you want the table restored.

- **sourceSchema**
  The name of the schema that was previously backed up.

- **directory**
  Specifies the path to the directory containing the backup from which you want to restore your schema. 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 the schema.

- **validate**
A Boolean value that specifies whether to validate the schema backup before restoring from it:

- If validate is false, the restore proceeds without any pre-validation.
- If validate is true, the backup is validated before the restoration is started. If the validation check finds inconsistencies, the errors are reported to the user, and the schema is not restored. If the inconsistencies are minor, you can choose to re-run this procedure with validate set to false.

**Backup and Restore Compatibility**

Note that you can only use specific restore procedures with specific types of backups. For example, you can use the RESTORE-table procedure to restore from a backup created by BACKUP-table, but you cannot use RESTORE-table to restore from a backup created by BACKUP-database. The following table summarizes backup-restore compatibility:

<table>
<thead>
<tr>
<th>If you backed up with this procedure:</th>
<th>You can restore with these procedures:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCS_UTIL.SYSCS_BACKUP_DATABASE</td>
<td>SYCS_UTIL.SYSCS_RESTORE_DATABASE</td>
</tr>
<tr>
<td>SYSCS_UTIL.SYSCS_BACKUP_SCHEMA</td>
<td>SYCS_UTIL.SYSCS_RESTORE_SCHEMA,</td>
</tr>
<tr>
<td></td>
<td>SYCS_UTIL.SYSCS_RESTORE_TABLE</td>
</tr>
<tr>
<td>SYCS_UTIL.SYSCS_BACKUP_TABLE</td>
<td>SYCS_UTIL.SYSCS_RESTORE_TABLE</td>
</tr>
</tbody>
</table>

**Backing Up and Restoring Statistics**

Note that statistics are also backed up and restored as of version 2.7.0.1924 (August 5, 2019) or later of Splice Machine. This means that if you restore a backup created with 2.7.0.1924 or later and the statistics were accurate when the backup was done, you do not need to use analyze to generate fresh statistics for the table immediately after restoring it. If the statistics were not accurate, you can run analyze to refresh them.

If you've restored from a table or schema backup and aren't sure if statistics were restored, you can use the following query to determine if statistics are available, replacing <mySchemaName> and <myTableName> with the appropriate names:

```
SELECT * FROM SYSVW.SYSTABLESTATISTICS
WHERE schemaname='<mySchemaName>' and tablename='<myTableName>'
```

**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: Backup, and Restore a Schema
This example shows you how to back up a schema, then restore it, in these steps:

- Backing Up the Schema
- Examining the Backup
- Restoring the Backup

Backing Up the Schema
This command line performs a full backup of the TPCH1 schema to the /backup directory on HDFS:

```
splice> CALL SYSCS_UTIL.SYSCS_BACKUP_SCHEMA('TPCH1', '/backup', 'full');
Success
```
FULL backup to /backup
1 row selected

See the reference page for the `SYSCS_UTIL.SYSCS_BACKUP_SCHEMA` system procedure for more information about backing up a schema.

Examining the Backup
After the backup completes, you can examine the `sys.sysbackup` table to find the ID of your new backup:

```
splice> SELECT * FROM sys.sysbackup;
BACKUP_ID |BEGIN_TIMESTAMP       |END_TIMESTAMP       |STATUS     |SCOP       |INCREMENTAL_PARENT_9 |INCREMENTAL_PARENT_9 |INCREMENTAL_PARENT_9 |INCREMENTAL_PARENT_9 |
-----------|----------------------|---------------------|------------|-----------|----------------------|---------------------|---------------------|---------------------|
125953     |2018-10-26 00:12:33.896|2018-10-26 00:42:53.546|SUCCESS    |SCHEM      |false                |false                |false                |
```

You can use the ID of your backup job to examine the `sys.sysbackupitems` and verify that the base table and two indexes have been backed up:

```
splice> SELECT * FROM sys.sysbackupitems;
```

SYSCS_UTIL.SYSCS_RESTORE_SCHEMA
**Restoring the Backup**

You can restore the schema to another schema on the same cluster, or on a different cluster. You can optionally specify that you want the backup validated before it is restored; the validation process checks for inconsistencies and missing files.

This command first validates the backed-up schema, and then restores it to a different schema named `NEWTPCH1`:

```sql
splice> CALL SYSCS_UTIL.SYSCS_RESTORE_SCHEMA('NEWTPCH1', 'TPCH1', '/backup', 125953, true);
Statement executed.
```

**See Also**

- [SYSCS_UTIL.SYSCS_BACKUP_SCHEMA](#)
- [SYSCS_UTIL.SYSCS_BACKUP_TABLE](#)
- [SYSCS_UTIL.SYSCS_RESTORE_TABLE](#)
- [SYSBACKUP](#)
- [SYSBACKUPITEMS](#)
- [SYSCS_UTIL.SYSCS_BACKUP_DATABASE](#)
- *Backing Up and Restoring Databases*
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**

```sql
SYCS_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 SYCS_UTIL.RESTORE_SNAPSHOT( 'snap_myschema_070417a' );
Statement executed.
```
SYCS_UTIL.SYCS_RESTORE_TABLE

The SYCS_UTIL.SYCS_RESTORE_TABLE system procedure restores a table that was previously backed up with the SYCS_UTIL.SYCS_BACKUP_TABLE procedure. You can restore the table to another table on the same cluster, or on a different cluster. The table and its indexes are created anew in your database.

**ENTERPRISE ONLY:** This feature is available only for the Splice Machine Enterprise version of our On-Premise Database product; contact Splice Machine Sales for information.

The table to which you are restoring must not already exist in the database; if it does, RESTORE_TABLE will not proceed.

**Syntax**

```
SYCS_UTIL.SYCS_RESTORE_TABLE( VARCHAR destSchema,
                                VARCHAR destTable,
                                VARCHAR sourceSchema,
                                VARCHAR sourceTable,
                                VARCHAR directory,
                                BIGINT backupId,
                                BOOLEAN validate );
```

**destSchema**
The name of the schema to which you want the table restored.

**destTable**
The name of the restored table.

**sourceSchema**
The name of the schema from which the table was backed up.

**sourceTable**
The name of the table that was backed up.

**directory**
Specifies the path to the directory containing the backup from which you want to restore your table. 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 the table.

validate

A Boolean value that specifies whether to validate the table backup before restoring from it:

- If `validate` is `false`, the restore proceeds without any pre-validation.
- If `validate` is `true`, the backup is validated before the restoration is started. (See `SYCS_UTIL.VALIDATE_TABLE_BACKUP`). If the validation check finds inconsistencies, the errors are reported to the user, and the table is *not* restored. If the inconsistencies are minor, you can choose to re-run this procedure with `validate` set to `false`.

**Results**

This procedure does not return a result.

**Backup and Restore Compatibility**

Note that you can only use specific restore procedures with specific types of backups. For example, you can use the `RESTORE_TABLE` procedure to restore from a backup created by `BACKUP_TABLE`, but you cannot use `RESTORE_TABLE` to restore from a backup created by `BACKUP_DATABASE`. The following table summarizes backup-restore compatibility:

<table>
<thead>
<tr>
<th>If you backed up with this procedure:</th>
<th>You can restore with these procedures:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SYCS_UTIL.SYSCS_BACKUP_DATABASE</code></td>
<td><code>SYCS_UTIL.SYSCS_RESTORE_DATABASE</code></td>
</tr>
<tr>
<td><code>SYCS_UTIL.SYSCS_BACKUP_SCHEMA</code></td>
<td><code>SYCS_UTIL.SYSCS_RESTORE_SCHEMA,</code></td>
</tr>
<tr>
<td></td>
<td><code>SYCS_UTIL.SYSCS_RESTORE_TABLE</code></td>
</tr>
<tr>
<td><code>SYCS_UTIL.SYSCS_BACKUP_TABLE</code></td>
<td><code>SYCS_UTIL.SYSCS_RESTORE_TABLE</code></td>
</tr>
</tbody>
</table>

**Backing Up and Restoring Statistics**

Note that statistics are also backed up and restored as of version 2.7.0.1924 (August 5, 2019) or later of Splice Machine. This means that if you restore a backup created with 2.7.0.1924 or later and the statistics were accurate when the backup was done, you do not need to use `analyze` to generate fresh statistics for the table immediately after restoring it. If the statistics were not accurate, you can run `analyze` to refresh them.

If you've restored from a table or schema backup and aren't sure if statistics were restored, you can use the following query to determine if statistics are available, replacing `<mySchemaName>` and `<myTableName>` with the appropriate names:
SELECT * FROM SYSVW.SYSTABLESTATISTICS
WHERE schemaname='<mySchemaName>' and tablename='<myTableName>'

**Backing Up and Restoring Statistics**

Note that statistics are also backed up and restored as of version 2.7.0.1924 (August 5, 2019) or later of Splice Machine. This means that if you restore a backup created with 2.7.0.1924 or later and the statistics were accurate when the backup was done, you do not need to use `analyze` to generate fresh statistics for the table immediately after restoring it. If the statistics were not accurate, you can run `analyze` to refresh them.

If you've restored from a table or schema backup and aren't sure if statistics were restored, you can use the following query to determine if statistics are available, replacing `<mySchemaName>` and `<myTableName>` with the appropriate names:

```
SELECT * FROM SYSVW.SYSTABLESTATISTICS
WHERE schemaname='<mySchemaName>' and tablename='<myTableName>'
```

**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: Backup, Validate, and Restore a Table**

This example shows you how to back up a table, then validate and restore it, in these steps:

- **Backing Up the Table**
- **Examining the Backup**
- **Validating the Backup**
- **Restoring the Backup**

**Backing Up the Table**

This command line performs a full backup of the TPCH100 LINEITEM table to the /backup directory on HDFS:

```
splice> CALL SYSCS_UTIL.SYSCS_BACKUP_TABLE('TPCH100', 'LINEITEM', '/backup', 'full');
Success
----------------------
FULL backup to /backup
1 row selected
```

See the reference page for the `SYSCS_UTIL.SYSCSBACKUP_TABLE` system procedure for more information about backing up a table.
**Examining the Backup**

After the backup completes, you can examine the `sys.sysbackup` table to find the ID of our new backup:

```sql
splice> SELECT * FROM sys.sysbackup;
BACKUP_ID       |BEGIN_TIMESTAMP          |END_TIMESTAMP            |STATUS     |SCOP
E               |INCR&|INCREMENTAL_PARENT_&|BACKUP_ITEM
----------------------------------------------------------------------------------------
-------------------------------------------
587516417      |2018-09-25 00:12:33.896  |2018-09-25 00:42:53.546  |SUCCESS    |TABL
E               |false|                    |3
```

You can use the ID of your backup job to examine the `sys.sysbackupitems` and verify that the base table and two indexes have been backed up:

```sql
splice> SELECT * FROM sys.sysbackupitems WHERE backup_Id=587516417 ;
BACKUP_ID   |ITEM             |BEGIN_TIMESTAMP           |END_TIMESTAMP
----------------------------------------------------------------------------------------
-            |
587516417   |splice:292000    |2018-09-25 00:12:40.512   |2018-09-25 00:32:14.856
587516417   |splice:292033    |2018-09-25 00:12:40.513   |2018-09-25 00:42:48.573
587516417   |splice:292017    |2018-09-25 00:12:40.512   |2018-09-25 00:41:25.683
3 rows selected
```

**Validating the Backup**

Before restoring the table, you can validate the backup:

```sql
splice> CALL SYSCS_UTIL.VALIDATE_TABLE_BACKUP( 'TPCH100', 'LINEITEM', '/backup', 587516417 );
Results
-------------------------------------------
-----
No corruptions found for backup.
1 row selected
```

See the reference page for the `SYSCS_UTIL.VALIDATE_TABLE_BACKUP` system procedure for more information about backup validation.

**Restoring the Backup**

You can restore the table to another table on the same cluster, or on a different cluster.

This command restores the backed-up table to table named `LINEITEM` in the `SPLICE` schema:
splice> CALL SYSCS_UTIL.SYSCS_RESTORE_TABLE('SPLICE', 'LINEITEM', 'TPCH100', 'LINEITEM', '/backup', 587516417, false);
Statement executed.

See Also

» SYSCS_UTIL.SYSCS_BACKUP_SCHEMA
» SYSCS_UTIL.SYSCS_BACKUP_TABLE
» SYSCS_UTIL.SYSCS_RESTORE_SCHEMA
» SYSCS_UTIL.SYSCS_VALIDATE_TABLE_BACKUP
» SYSBACKUP
» SYSBACKUPITEMS
» SYSCS_UTIL.SYSCS_Backup_DATABASE
» Backing Up and Restoring Databases
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.

Modifying global database property values can have negative and potentially disastrous impact on how your database performs. You should only use this procedure if you truly understand the consequences of the change(s) you are making.

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.deadlockTimeout', '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.SYCS_SET_LOGGER_LEVEL( 'com.splicemachine.mrio.api', 'DEBUG' );
Statement executed.

See Also

- SYCS_UTIL.SYCS_GET_LOGGERS
- SYCS_UTIL.SYCS_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**

```java
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:

```sql
CALL SYSCS_UTIL.SET_PURGE_DELETED_ROWS('SPLICE', 'my_table', true);
```
SYSCS_UTIL.SET_STATS_EXTRAPOLATION_FOR_COLUMN

Use the SYSCS_UTIL.SET_STATS_EXTRAPOLATION_FOR_COLUMN system procedure to specify whether or not you want statistics extrapolation used for a specific column in a table.

About Statistics Extrapolation

Statistics extrapolation allows the Splice Machine optimizer to extrapolate statistics for specific column values that have not yet been analyzed. This means that the optimizer can then estimate values for these columns in rows that have been added to a table since the most recent ANALYZE operation was run, which can yield better query plans.

With extrapolation enabled, the optimizer captures the growth pattern for each eligible column and then uses that pattern to create an algorithm for extrapolating missing values.

Extrapolation is only supported for columns with the following data types:

- TINYINT
- SMALLINT
- INTEGER
- LONGINT
- REAL
- DOUBLE
- DECIMAL
- DATE
- TIMESTAMP

Syntax

SYCS_UTIL.SYSCS_SET_LOGGER_LEVEL( VARCHAR schema,
                                      VARCHAR table,
                                      VARCHAR column,
                                      SMALLINT useExtrapolation )

schemaName
   Specifies the schema of the table.

tableName
   Specifies the name of the table.

column
   Specifies the name of the column for which you are setting the useExtrapolation value.
**useExtrapolation**

Specifies whether you want extrapolation enabled (value=1) or disabled (value=0 or NULL) for the column. This value defaults to disabled.

**Results**

This procedure does not return a result.

**Example**

This example enables extrapolation for column A5 in myTable:

```sql
CREATE TABLE myTable (a5 INT, b5 DECIMAL(10,2), c5 DATE, d5 TIMESTAMP, e5 VARCHAR(10), a55 INT, c55 DATE);
CALL SYSCS_UTIL.SET_STATS_EXTRAPOLATION_FOR_COLUMN('SPLICE', 'myTable', 'A5', 1);
```

And this call disables extrapolation for the same column:

```sql
CALL SYSCS_UTIL.SET_STATS_EXTRAPOLATION_FOR_COLUMN('SPLICE', 'myTable', 'A5', 0);
```
SYCS_UTIL.SHOW_CREATE_TABLE

The SYCS_UTIL.SHOW_CREATE_TABLE system procedure returns the DDL to create a table, in string form.

Syntax

```
SYCS_UTIL.SHOW_CREATE_TABLE( VARCHAR schemaName,
                               VARCHAR tableName );
```

| schemaName | The name of the table's schema. |
| tableName  | The name of the table.          |

Results

This procedure returns a string representation of the `CREATE TABLE` statement DDL to create the specified table.
Example

splice> CREATE TABLE Players(
    ID        SMALLINT NOT NULL,
    Team      VARCHAR(64) NOT NULL,
    Name      VARCHAR(64) NOT NULL,
    Position  CHAR(2),
    DisplayName VARCHAR(24),
    BirthDate DATE
);
0 rows inserted/updated/deleted

CALL SYSCS_UTIL.SHOW_CREATE_TABLE( 'SPLICE', 'PLAYERS' );

DDL
----------------------------------------------------
CREATE TABLE "SPLICE"."PLAYERS" ( 
    "ID" SMALLINT NOT NULL ,
    "TEAM" VARCHAR(64) NOT NULL ,
    "NAME" VARCHAR(64) NOT NULL ,
    "POSITION" CHAR(2) ,
    "DISPLAYNAME" VARCHAR(24) ,
    "BIRTHDATE" DATE
 ) ;

1 row selected

See Also

» The show create table command uses this procedure to generate its output.
**SYSCS_UTIL.SNAPSHOT_SCHEMA**

The SYSCS_UTIL.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

```sql
SYCS_UTIL.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*:

```sql
splice> CALL SYCS_UTIL.SNAPSHOT_SCHEMA('mySchema', 'snap_myschema_070417a');
Statement executed.
```
SYSCS_UTIL.SNAPSHOT_TABLE

The SYSCS_UTIL.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**

```sql
SYSCS_UTIL.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_TABLE( 'mySchema', 'myTable', 'snap_myschema_070417a' );
Statement executed.
```
SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX

The SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX system procedure pre-splits a table or index that you are importing 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 parameter values that you pass into this procedure should match the values for the same-named parameters that you use when you subsequently call the SYSCS_UTIL.BULK_IMPORT_HFILE procedure to perform the import.

This table includes a brief description of each parameter; additional information is available in the Ingestion Parameter Values topic of our Importing Data tutorial.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaName</td>
<td>The name of the schema of the table into which to import.</td>
<td>SPLICE</td>
</tr>
<tr>
<td>tableName</td>
<td>The name of the table into which to import.</td>
<td>playerTeams</td>
</tr>
<tr>
<td>indexName</td>
<td>The name of the index into which to import.</td>
<td>playerTeamsIdx</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>columnList</td>
<td>A comma-separated list of the columns used for split keys.</td>
<td>See the Example below.</td>
</tr>
<tr>
<td>fileName</td>
<td>The name of the file in which you have specified the split keys.</td>
<td>/data/mydata/mytable.csv</td>
</tr>
<tr>
<td>columnDelimiter</td>
<td>The character used to separate columns. Specify null if using the comma (,) character as your delimiter.</td>
<td>'</td>
</tr>
<tr>
<td>characterDelimiter</td>
<td>The character used to delimit strings in the imported data.</td>
<td>'&quot;', '''</td>
</tr>
<tr>
<td>timestampFormat</td>
<td>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 Java.sql.Timestamp default format, which is: &quot;yyyyMMdd-HHmm:ssSSZ&quot;.</td>
<td>'yyyyMMdd-MM-dd HH:mm:ss.SSZ'</td>
</tr>
<tr>
<td>dateFormat</td>
<td>The format of dates 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: &quot;yyyyMMdd-MM-dd&quot;.</td>
<td>yyyy-MM-dd</td>
</tr>
<tr>
<td>timeFormat</td>
<td>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: &quot;HH:mm:ss&quot;.</td>
<td>HH:mm:ss</td>
</tr>
<tr>
<td>maxBadRecords</td>
<td>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.</td>
<td>25</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>badRecordDirectory</td>
<td>The directory in which bad record information is logged. Splice Machine logs information to the <code>&lt;import_file_name&gt;.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>. On a cluster, this directory <strong>MUST be on S3, HDFS (or MapR-FS)</strong>. If you're using our Database Service product, it must be on S3.</td>
<td>'importErrsDir'</td>
</tr>
<tr>
<td>oneLineRecords</td>
<td>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.</td>
<td>true</td>
</tr>
<tr>
<td>charset</td>
<td>The character encoding of the import file. The default value is UTF-8.</td>
<td>null</td>
</tr>
</tbody>
</table>

**Usage**

You can use the SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX procedure to pre-split a data file that you're importing with the SYSCS_UTIL.BULK_IMPORT_HFILE procedure. Alternatively, SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX can sample the data in your file and create the split keys itself.

When you pre-split your data, make sure that you set the skipSampling parameter to true when calling SYSCS_UTIL.BULK_IMPORT_HFILE; that tells the bulk import procedure that you have already split your data.

The Best Practices: Bulk Importing Flat Files section of our Importing Data Tutorial describes the different methods for using our bulk HFile import functionality.

**Example**

This example details the steps used to import data inHFile format by:

- specifying the split keys manually in a CSV file
- using SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX to pre-split the file you’re importing
- calling SYSCS_UTIL.BULK_IMPORT_HFILE to import the file

Follow these steps:
1. **Create a directory on HDFS for the import; for example:**

   ```bash
   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:**

   ```sql
   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_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)  
   );

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

3. **Determine the split row keys for your table and set up the pre-splits:**

   **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`, which will be specified as the value of the `fileName` parameter. Note that each of our three keys includes a second column that is null:

   ```plaintext
   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:

<table>
<thead>
<tr>
<th>Region 1</th>
<th>Region 2</th>
<th>Region 3</th>
<th>Region 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1500000</td>
<td>3000000</td>
<td>4500000</td>
</tr>
</tbody>
</table>

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

'\texttt{L\_ORDERKEY,L\_LINENUMBER}'

c. Invoke SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX to pre-split your table file:

```java
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 \texttt{shipDateIndex.csv}, which will be specified as the value of the fileName parameter. Note that each of our keys includes null column values:

<table>
<thead>
<tr>
<th>Key 1</th>
<th>Key 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994-01-01</td>
<td></td>
</tr>
<tr>
<td>1996-01-01</td>
<td></td>
</tr>
</tbody>
</table>

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

'\texttt{L\_SHIPDATE,L\_PARTKEY,L\_EXTENDEDPRICE,L\_DISCOUNT}'

d. Invoke SYSCS_UTIL.SYSCS_SPLIT_TABLE_OR_INDEX to pre-split your index file:
5. **Import the HFiles Into Your Database**

Once you have pre-split your table and indexes, call `SYSCS_UTIL.BULK_IMPORT_HFILE` to generate and import the HFiles into your Splice Machine database:

```sql
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**

- Best Practices: Ingestion
- `SYCS_UTIL.IMPORT_DATA`
- `SYCS_UTIL.MERGE_DATA_FROM_FILE`
- `SYCS_UTIL.BULK_IMPORT_HFILE`
This procedure splits a table or index file that you want to bulk import into HFiles, using the *split keys* that you specify. The split keys are specified in a CSV file that is encoded in HBase format.

By default, you should keep a copy of your previous CSV file (created by SYSCS_UTIL.COMPUTE_SPLIT_KEY) in the HBase format. However, if you do not have a previous copy of your split key CSV file, Splice Machine recommends using the SYSCS_UTIL.SYSCS_SPLIT_TABLE OR_INDEX system procedure instead of this one. The combination of using SYSCS_UTIL.COMPUTE_SPLIT_KEY with SYSCS_UTIL.SYSCS_SPLIT_TABLE OR_INDEX_AT_POINTS is exactly equivalent to using SYSCS_UTIL.SYSCS_SPLIT_TABLE OR_INDEX.

### Syntax

```sql
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 list of split points for the table or index, supplied in HBase format in a CSV file this list can be created by a previous call to the SYSCS_UTIL.COMPUTE_SPLIT_KEY procedure, or you can prepare it manually, in which case, it needs to follow the criteria specified in the next section, *Split Points CSV File Format*.

### Split Points CSV File Format

If you are manually preparing the *splitPoints* CSV file, you must create a version of the file you are importing that contains only rows that are region boundary rows. Each row in the file:

- contains only the primary key column value, if you’re importing a table.
- contains only the index column values, if you’re importing an index.
Usage
You can use the `SYCS_UTIL.SYCS_SPLIT_TABLE_OR_INDEX_AT_POINTS` procedure to pre-split a data file that you're importing with the `SYCS_UTIL.BULK_IMPORT_HFILE` procedure.

When you pre-split your data, make sure that you set the `skipSampling` parameter to `true` when calling `SYCS_UTIL.BULK_IMPORT_HFILE`; that tells the bulk import procedure that you have already split your data.

The Best Practices: Bulk Importing Flat Files section of our *Importing Data Tutorial* describes the different methods for using our bulk HFile import functionality.

See Also

» Best Practices: Ingestion
» `SYCS_UTIL.IMPORT_DATA`
» `SYCS_UTIL.MERGE_DATA_FROM_FILE`
» `SYCS_UTIL.BULK_IMPORT_HFILE`
» `SYCS_UTIL.COMPUTE_SPLIT_KEY`
» `SYCS_UTIL.SYCS_SPLIT_TABLE_OR_INDEX_AT_POINTS`
» `SYCS_UTIL.SYCS_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 updates 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:

- **Implementation**, which is compiled Java byte code that is stored in the Splice jar and is included in the CLASSPATH of the Splice server.

- **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

» SYCS_UTIL SYSCS_UPDATE_SYSTEM_PROCEDURE
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 `SYCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE` procedure whenever you update your Splice Machine software installation.

If you have called the `SYCS_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

```
SYCS_UPDATE_METADATA_STORED_STATEMENTS()
```

### Results

This procedure does not return a result.

### Example

```
splice> CALL SYCS_UPDATE_METADATA_STORED_STATEMENTS();
Statement executed.
```

### See Also

- `SYCS_UTIL.SYSCS_EMPTY_STATEMENT_CACHE`
The `SYSCS_UTIL.SYSCS_UPDATE_SCHEMA_OWNER` system procedure changes the owner of a schema.

**Syntax**

```sql
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**

```sql
splice> CALL SYSCS_UTIL.SYSCS_UPDATE_SCHEMA_OWNER( 'SPLICEBBALL', 'Walt');
Statement executed.
```
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**

» SYCS_UTIL.SYSCS_UPDATE_ALL_SYSTEM_PROCEDURES
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.

This procedure has been deprecated; you can still use it in the current release, but Splice Machine strongly recommends using the more performant SYSCS_UTIL.MERGE_DATA_FROM_FILE system procedure instead.

After a successful import completes, a simple report displays, showing how many files were imported, and how many record imports succeeded or failed.

This procedure is one of several built-in system procedures provided by Splice Machine for importing data into your database. See our Best Practices: Ingestions chapter for help with selecting the right process for your situation.

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 Ingestion Parameter Values.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaName</td>
<td>The name of the schema of the table into which to import.</td>
<td>SPLICE</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>tableName</td>
<td>The name of the table into which to import.</td>
<td>playerTeams</td>
</tr>
<tr>
<td>insertColumnList</td>
<td>The names, in single quotes, of the columns to import. If this is null, all columns are imported.</td>
<td>'ID, TEAM'</td>
</tr>
<tr>
<td>fileOrDirectoryName</td>
<td>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. 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. See the Configuring an S3 Bucket for Splice Machine Access topic for information about accessing data on S3.</td>
<td>/data/mydata/mytable.csv 's3a://splice-benchmark-data/flat/TPCH/100/region'</td>
</tr>
<tr>
<td>columnDelimiter</td>
<td>The character used to separate columns. Specify null if using the comma (,) character as your delimiter.</td>
<td>'</td>
</tr>
<tr>
<td>characterDelimiter</td>
<td>The character used to delimit strings in the imported data.</td>
<td>' '' ' ',' ' '</td>
</tr>
<tr>
<td>timestampFormat</td>
<td>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 Java.sql.Timestamp default format, which is: &quot;yyyy-MM-dd HH:mm:ss&quot;. All of the timestamps in the file you are importing must use the same format.</td>
<td>'yyyy-MM-dd HH:mm:ss.SSZ'</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>dateFormat</td>
<td>The format of datestamps 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: &quot;yyyy-MM-dd&quot;.</td>
<td>yyyy-MM-dd</td>
</tr>
<tr>
<td>timeFormat</td>
<td>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: &quot;HH:mm:ss&quot;.</td>
<td>HH:mm:ss</td>
</tr>
<tr>
<td>badRecordsAllowed</td>
<td>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.</td>
<td>25</td>
</tr>
<tr>
<td>badRecordDirectory</td>
<td>The directory in which bad record information is logged. Splice Machine logs information to the (&lt;import_file_name&gt;.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.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td>'importErrsDir'</td>
</tr>
<tr>
<td>oneLineRecords</td>
<td>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.</td>
<td>true</td>
</tr>
<tr>
<td>charset</td>
<td>The character encoding of the import file. The default value is UTF-8.</td>
<td>null</td>
</tr>
</tbody>
</table>

**Results**

SYCS_UTIL.UPSERT_DATA_FROM_FILE displays a summary of the import process results that looks like this:

```
<table>
<thead>
<tr>
<th>rowsImported</th>
<th>failedRows</th>
<th>files</th>
<th>dataSize</th>
<th>failedLog</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>0</td>
<td>1</td>
<td>4720</td>
<td>NONE</td>
</tr>
</tbody>
</table>
```

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.UPSERT_DATA_FROM_FILE` from the similar `SYSCS_UTIL.IMPORT_DATA` 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 Ingestion Parameters 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 Best Practices: Importing Flat Files topic contains a more extensive set of examples.

Example 1: Updating our doc examples player data
This example shows the \texttt{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, 0, null, true, null);
```

```
rowsImported | failedRows | files | dataSize | failedLo
-------------|------------|------|----------|----------
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( 'SPLICE', 'MYTABLE', null, 'data.csv',
  '\t', null, null, null, null, 0, 'BAD', false, null);
```

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

```
SYSCS\_UTIL.UPSERT\_DATA\_FROM\_FILE( 'SPLICE', 'MYTABLE', null, 'data.csv',
  '\t', '\"', null, null, null, 0, 'BAD', false, null);
```

See Best Practices: Importing Flat Files for more examples.

See Also

» Best Practices - Ingestion
Importing Data Examples

- SYSCS_UTIL.IMPORT_DATA
- SYSCS_UTIL.MERGE_DATA_FROM_FILE
**SYSCS_UTIL.VACUUM**

When you drop a table from your database, Splice Machine marks the space occupied by the table as *deleted*, but does not actually free the physical space. That space is only reclaimed when you call the `SYSCS_UTIL.VACUUM` system procedure, which does the following:

1. Waits for all previous transactions to complete (and times out if this takes too long).
2. Gets a list of all of the HBase tables in use.
3. Compares that list with a list of objects currently in use in your database, and deletes any HBase tables that are no longer in use in your database.

This is a synchronous operation; when it completes, you'll see the following message:

```
Statement executed.
```

If you see an exception message instead of the completion message, please try calling `SYSCS_UTIL.VACUUM` once again.

**Syntax**

```
SYSCS_UTIL.VACUUM()
```

**Usage**

To call `SYSCS_UTIL.VACUUM`, you must have execute permission on the (internal-only, undocumented) `SYSCS_UTIL.SYSCS_GET_OLDEST_ACTIVE_TRANSACTION` system procedure. You can use the following command to grant permission, replacing `myUserId` with your user ID:

```
grant EXECUTE on procedure SYSCS_UTIL.SYSCS_GET_OLDEST_ACTIVE_TRANSACTION to myUserId;
```

**Example**

```
splice> CALL SYSCS_UTIL.VACUUM();
Ready to accept connections.
```
The `SYCS_UTIL.VALIDATE_BACKUP` system procedure validates a database backup by checking for inconsistencies; it reports on missing files and bad checksum values.

For more information, see the *Backing Up and Restoring* topic.

**Syntax**

```
SYCS_UTIL.VALIDATE_BACKUP( VARCHAR backupDir,
                          BIGINT backupId );
```

*backupDir*

Specifies the path to the directory containing the backup you want to validate. 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.

*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.

**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**

This section contains an example showing a successful validation and a validation that reports errors.
Example 1: Successful Validation

splice> SELECT * FROM SYS.SYSBACKUP;
<table>
<thead>
<tr>
<th>BACKUP_ID</th>
<th>BEGIN_TIMESTAMP</th>
<th>END_TIMESTAMP</th>
<th>STATUS</th>
<th>FILESYSTEM</th>
</tr>
</thead>
</table>
2 rows selected

splice> CALL SYSCS_UTIL.SYSCS_VALIDATE_BACKUP('./dbBackups/', 74101);
Statement executed.

Example 2: Validation Failure

splice> SELECT * FROM SYS.SYSBACKUP;
<table>
<thead>
<tr>
<th>BACKUP_ID</th>
<th>BEGIN_TIMESTAMP</th>
<th>END_TIMESTAMP</th>
<th>STATUS</th>
<th>FILESYSTEM</th>
</tr>
</thead>
</table>
2 rows selected

splice> CALL SYSCS_UTIL.SYSCS_VALIDATE_DATABASE('./dbBackups/', 63541);
Results | Warnings
--------|--------
BR010 | A data file ./dbBackups/BACKUP_63541/tables/SPLICE_TXN/f4460c47f6c96fe8d76c0def11e22dc8/V/c7350dec1caaf4a11a561472675edaldd is missing. The restored table may be corrupted.
1 row selected

See Also

- Backing Up and Restoring Databases
- SYSCS_UTIL.SYCS_BACKUP_DATABASE
- SYSCS_UTIL.SYSCS_DELETE_BACKUP
» SYSCS_UTIL.SYSCS_DELETE_OLD_BACKUPS
» SYSCS_UTIL.VALIDATE_BACKUP
» SYSBACKUP
» SYSBACKUPITEMS
SYSCS_UTIL.VALIDATE_TABLE_BACKUP

The SYSCS_UTIL.VALIDATE_TABLE_BACKUP system procedure validates a table backup by checking for inconsistencies; it reports on missing files and bad checksum values.

Syntax
SYCS_UTIL.VALIDATE_TABLE_BACKUP( VARCHAR schemaName,
                                  VARCHAR tableName,
                                  VARCHAR directory,
                                  BIGINT backupId,
                                  
  
  **schemaName**
  The name of the table's schema.

  **tableName**
  The name of the table whose backup you want to validate.

  **directory**
  Specifies the path to the directory containing the backup you want to validate. 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.

  **backupId**
  The ID of the table backup job from which you want to restore your table.

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: Backup, Validate, and Restore a Table
This example shows you how to back up a table, then validate and restore it, in these steps:

- Backing Up the Table
- Examining the Backup
Validating the Backup

Restoring the Backup

**Backing Up the Table**

This command line performs a full backup of the TPCH100 LINEITEM table to the /backup directory on HDFS:

```sql
splice> CALL SYSCS_UTIL.SYSCS_BACKUP_TABLE('TPCH100', 'LINEITEM', '/backup', 'full');
Success
FULL backup to /backup
```

1 row selected

See the reference page for the `SYSCS_UTIL.SYSCSBACKUP_TABLE` system procedure for more information about backing up a table.

**Examining the Backup**

After the backup completes, you can examine the `sys.sysbackup` table to find the ID of our new backup:

```sql
splice> SELECT * FROM sys.sysbackup;
```

<table>
<thead>
<tr>
<th>BACKUP_ID</th>
<th>BEGIN_TIMESTAMP</th>
<th>END_TIMESTAMP</th>
<th>STATUS</th>
<th>SCOPE</th>
<th>INC</th>
</tr>
</thead>
<tbody>
<tr>
<td>587516417</td>
<td>2018-09-25 00:12:33.896</td>
<td>2018-09-25 00:42:53.546</td>
<td>SUCCESS</td>
<td>TABLE</td>
<td>fal</td>
</tr>
<tr>
<td>-1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

You can use the ID of your backup job to examine the `sys.sysbackupitems` and verify that the base table and two indexes have been backed up:

```sql
splice> SELECT * FROM sys.sysbackupitems WHERE backup_Id=587516417 ;
```

<table>
<thead>
<tr>
<th>BACKUP_ID</th>
<th>ITEM</th>
<th>BEGIN_TIMESTAMP</th>
<th>END_TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>587516417</td>
<td>splice:292000</td>
<td>2018-09-25 00:12:40.512</td>
<td>2018-09-25 00:32:14.856</td>
</tr>
<tr>
<td>587516417</td>
<td>splice:292033</td>
<td>2018-09-25 00:12:40.513</td>
<td>2018-09-25 00:42:48.573</td>
</tr>
<tr>
<td>587516417</td>
<td>splice:292017</td>
<td>2018-09-25 00:12:40.512</td>
<td>2018-09-25 00:41:25.683</td>
</tr>
</tbody>
</table>

3 rows selected

**Validating the Backup**

Before restoring the table, you can validate the backup:
splice> CALL SYSCS_UTIL.VALIDATE_TABLE_BACKUP( 'TPCH100', 'LINEITEM', '/backup', 587516417 );
Results

No corruptions found for backup.

1 row selected

**Restoring the Backup**

You can restore the table to another table on the same cluster, or on a different cluster.

This command restores the backed-up table to table named LINEITEM in the SPLICE schema:

splice> CALL SYSCS_UTIL.SYSCS_RESTORE_TABLE('SPLICE', 'LINEITEM', 'TPCH100', 'LINEITEM', '/backup', 587516417, false);
Statement executed.

See the reference page for the `SYSCS_UTIL.SYSCS_RESTORE_TABLE` system procedure for more information about restoring a backed-up table.

**See Also**

- `SYSCS_UTIL.SYSCS_BACKUP_TABLE`
- `SYSCS_UTIL.SYSCS_RESTORE_TABLE`
- `SYSBACKUP`
- `SYSBACKUPITEMS`
- `SYSCS_UTIL.SYSCS_BACKUP_DATABASE`
- *Backing Up and Restoring Databases*
# 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

The following table lists the System Tables with backups information:

The tables listed in this section apply only to our on-premise database product.

<table>
<thead>
<tr>
<th>System Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSBACKUP</td>
<td>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.</td>
</tr>
<tr>
<td>SYSBACKUPITEMS</td>
<td>Information about the items backed up for each backup job.</td>
</tr>
</tbody>
</table>

## Database Objects Tables

These are the System Tables with information about database objects:

<table>
<thead>
<tr>
<th>System Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSALIASES</td>
<td>Describes the procedures, functions, and user-defined types in the database.</td>
</tr>
<tr>
<td>SYSCHECKS</td>
<td>Describes the check constraints within the current database.</td>
</tr>
</tbody>
</table>
### System Tables

<table>
<thead>
<tr>
<th>System Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYScolumns</td>
<td>Describes the columns within all tables in the current database.</td>
</tr>
<tr>
<td>SYSCONSTRAINTS</td>
<td>Describes the information common to all types of constraints within the current database.</td>
</tr>
<tr>
<td>SYSDEPENDS</td>
<td>Stores the dependency relationships between persistent objects in the database.</td>
</tr>
<tr>
<td>SYSFOREIGNKEYS</td>
<td>Describes the information specific to foreign key constraints in the current database.</td>
</tr>
<tr>
<td>SYSKEYS</td>
<td>Describes the specific information for primary key and unique constraints within the current database.</td>
</tr>
<tr>
<td>SYSROLES</td>
<td>Stores the roles in the database.</td>
</tr>
<tr>
<td>SYSSCHEMAS</td>
<td>Describes the schemas within the current database.</td>
</tr>
<tr>
<td>SYSSEQUENCES</td>
<td>Describes the sequence generators in the database.</td>
</tr>
<tr>
<td>SYSSNAPSHOTS</td>
<td>Stores metadata for a Splice Machine snapshot.</td>
</tr>
<tr>
<td>SYSTABLES</td>
<td>Describes the tables and views within the current database.</td>
</tr>
<tr>
<td>SYSTRIGGERS</td>
<td>Describes the triggers defined for the database.</td>
</tr>
<tr>
<td>SYSVIEWS</td>
<td>Describes the view definitions within the current database.</td>
</tr>
</tbody>
</table>

### Database Permissions Tables

These are the System Tables with database permissions information:

<table>
<thead>
<tr>
<th>System Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCOLPERMS</td>
<td>Stores the column permissions that have been granted but not revoked.</td>
</tr>
<tr>
<td>SYSPERMS</td>
<td>Describes the usage permissions for sequence generators and user-defined types.</td>
</tr>
<tr>
<td>SYSROUTINEPERMS</td>
<td>Stores the permissions that have been granted to routines.</td>
</tr>
<tr>
<td>SYSTABLEPERMS</td>
<td>Stores the table permissions that have been granted but not revoked.</td>
</tr>
</tbody>
</table>

### Database Statistics Tables

These are the System Tables and Views with database statistics information:
<table>
<thead>
<tr>
<th>System Table/View</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCOLUMNSTATISTICS</td>
<td>Statistics gathered for each column in each table.</td>
</tr>
<tr>
<td>SYSTABLESTATISTICS</td>
<td>Describes the statistics for each table within the current database.</td>
</tr>
</tbody>
</table>

**System Information Tables**

These are the System Tables with system information:

<table>
<thead>
<tr>
<th>System Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSCONGLOMERATES</td>
<td>Describes the conglomerates within the current database. A conglomerate is a unit of storage and is either a table or an index.</td>
</tr>
<tr>
<td>SYSFILES</td>
<td>Describes jar files stored in the database.</td>
</tr>
<tr>
<td>SYSSTATEMENTS</td>
<td>Describes the prepared statements in the database.</td>
</tr>
<tr>
<td>SYSUSERS</td>
<td>Stores user credentials when NATIVE authentication is enabled.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIASID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for the alias</td>
</tr>
<tr>
<td>ALIAS</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>Alias (in the case of a user-defined type, the name of the user-defined type)</td>
</tr>
<tr>
<td>SCHEMAID</td>
<td>CHAR</td>
<td>36</td>
<td>YES</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>JAVACLASSNAME</td>
<td>LONG VARCHAR</td>
<td>2,147,483,647</td>
<td>NO</td>
<td>The Java class name</td>
</tr>
<tr>
<td>ALIASTYPE</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>'F' (function), 'P' (procedure), 'A' (user-defined type)</td>
</tr>
<tr>
<td>NAMESPACE</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>'F' (function), 'P' (procedure), 'A' (user-defined type)</td>
</tr>
<tr>
<td>SYSTEMALIAS</td>
<td>BOOLEAN</td>
<td>1</td>
<td>NO</td>
<td>YES (system supplied or built-in alias) \ NO (alias created by a user)</td>
</tr>
<tr>
<td>ALIASINFO</td>
<td>org.apache.Splice Machine. catalog.AliasInfo</td>
<td>-1</td>
<td>YES</td>
<td>A Java interface that encapsulates the additional information that is specific to an alias</td>
</tr>
<tr>
<td>SPECIFICNAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>System-generated identifier</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKUP_ID</td>
<td>BIGINT</td>
<td>19</td>
<td>NO</td>
<td>The backup ID</td>
</tr>
<tr>
<td>BEGIN_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>29</td>
<td>NO</td>
<td>The start time of the backup</td>
</tr>
<tr>
<td>END_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>29</td>
<td>YES</td>
<td>The end time of the backup</td>
</tr>
<tr>
<td>STATUS</td>
<td>VARCHAR</td>
<td>10</td>
<td>NO</td>
<td>The status of the backup</td>
</tr>
<tr>
<td>FILESYSTEM</td>
<td>VARCHAR</td>
<td>32642</td>
<td>NO</td>
<td>The backup destination directory</td>
</tr>
<tr>
<td>SCOPE</td>
<td>VARCHAR</td>
<td>10</td>
<td>NO</td>
<td>The scope of the backup: database, schemas, tables, etc. The current allowable values are: D for the entire database</td>
</tr>
<tr>
<td>INCREMENTAL_BACKUP</td>
<td>BOOLEAN</td>
<td>1</td>
<td>NO</td>
<td>YES for incremental backups, NO for full backups</td>
</tr>
<tr>
<td>INCREMENTAL_PARENT_BACKUP_ID</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>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.</td>
</tr>
<tr>
<td>BACKUP_ITEM</td>
<td>INTEGER</td>
<td>10</td>
<td>YES</td>
<td>The number of tables that were backed up.</td>
</tr>
</tbody>
</table>

**NOTE:** Incremental backups are not yet available.
SYSBACKUPITEMS System Table

The SYSBACKUPITEMS table maintains information about each item (table) backed up during a backup.

**SYSBACKUPITEMS system table**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKUP_ID</td>
<td>BIGINT</td>
<td>19</td>
<td>NO</td>
<td>The backup ID.</td>
</tr>
<tr>
<td>ITEM</td>
<td>VARCHAR</td>
<td>32642</td>
<td>NO</td>
<td>The name of the item.</td>
</tr>
<tr>
<td>BEGIN_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>29</td>
<td>NO</td>
<td>The start time of backing up this item.</td>
</tr>
<tr>
<td>END_TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>29</td>
<td>YES</td>
<td>The end time of backing up this item.</td>
</tr>
<tr>
<td>SNAPSHOT_NAME</td>
<td>VARCHAR</td>
<td>32642</td>
<td>NO</td>
<td>The name of the snapshot associated with this item.</td>
</tr>
</tbody>
</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

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINTID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for the constraint</td>
</tr>
<tr>
<td>CHECKDEFINITION</td>
<td>LONG VARCHAR</td>
<td>32,700</td>
<td>NO</td>
<td>Text of check constraint definition</td>
</tr>
<tr>
<td>REFERENCEDCOLUMNS</td>
<td>com.splicemachine.db.catalog.ReferencedColumns</td>
<td>-1</td>
<td>NO</td>
<td>Description of the columns referenced by the check constraint</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLPERMSID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Used by the dependency manager to track the dependency of a view, trigger, or constraint on the column level permissions</td>
</tr>
<tr>
<td>GRANTEE</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The authorization ID of the user or role to which the privilege was granted</td>
</tr>
<tr>
<td>GRANTOR</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The authorization ID of the user who granted the privilege. Privileges can be granted only by the object owner</td>
</tr>
<tr>
<td>TABLEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>The unique identifier for the table on which the permissions have been granted</td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Length</td>
<td>Nullable</td>
<td>Contents</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>--------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TYPE</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>If the privilege is non-grantable, the valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>☞ 's' for SELECT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>☞ 'u' for UPDATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>☞ 'r' for REFERENCES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If the privilege is grantable, the valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>☞ 'S' for SELECT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>☞ 'U' for UPDATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>☞ 'R' for REFERENCES</td>
</tr>
<tr>
<td>COLUMNS</td>
<td>org.apache.SpliceMachine.iapi.services.io.FormatableBitSet</td>
<td>-1</td>
<td>NO</td>
<td>A list of columns to which the privilege applies</td>
</tr>
</tbody>
</table>

This class is not part of the public API.

**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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFERENCEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Identifier for table (join with SYSTABLES.TABLEID)</td>
</tr>
<tr>
<td>COLUMNNAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>Column or parameter name</td>
</tr>
<tr>
<td>COLUMNNUMBER</td>
<td>INTEGER</td>
<td>10</td>
<td>NO</td>
<td>The position of the column within the table</td>
</tr>
<tr>
<td>COLUMNDATATYPE</td>
<td>com.splicemachine.db.catalog.TypeDescriptor</td>
<td>-1</td>
<td>NO</td>
<td>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 TypeDescriptor that refers to the appropriate type alias in SYS.SYSALIASES.</td>
</tr>
<tr>
<td>COLUMNDEFAULT</td>
<td>java.io.Serializable</td>
<td>-1</td>
<td>YES</td>
<td>For tables, describes default value of the column. The toString() 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.</td>
</tr>
<tr>
<td>COLUMNDEFAULTID</td>
<td>CHAR</td>
<td>36</td>
<td>YES</td>
<td>Unique identifier for the default value</td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Length</td>
<td>Nullable</td>
<td>Contents</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>--------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AUTOINCREMENTVALUE</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>What the next value for column will be, if the column is an identity column</td>
</tr>
<tr>
<td>AUTOINCREMENTSTART</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>Initial value of column (if specified), if it is an identity column</td>
</tr>
<tr>
<td>AUTOINCREMENTINC</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>Amount column value is automatically incremented (if specified), if the column is an identity column</td>
</tr>
<tr>
<td>COLLECTSTATS</td>
<td>BOOLEAN</td>
<td>1</td>
<td>YES</td>
<td>Whether or not to collect statistics on the column.</td>
</tr>
<tr>
<td>PARTITIONPOSITION</td>
<td>INTEGER</td>
<td>10</td>
<td>YES</td>
<td>This is used for external tables, to indicate the partitioning column position</td>
</tr>
<tr>
<td>USEEXTRAPOLATION</td>
<td>TINYINT</td>
<td>3</td>
<td>YES</td>
<td>Whether or not to use statistics extrapolation on this column. A value of 1 indicates that extrapolation is enabled for the column; a value of 0 or NULL indicates that extrapolation is disabled</td>
</tr>
</tbody>
</table>

**See Also**

- About System Tables
SYSCOLUMNSTATISTICS System View

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 view.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMANAME</td>
<td>VARCHAR</td>
<td>32672</td>
<td>YES</td>
<td>The name of the schema.</td>
</tr>
<tr>
<td>TABLENAME</td>
<td>VARCHAR</td>
<td>32672</td>
<td>YES</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>COLUMNNAME</td>
<td>VARCHAR</td>
<td>32672</td>
<td>YES</td>
<td>The name of the column.</td>
</tr>
<tr>
<td>CARDINALITY</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>The estimated number of distinct values for the column.</td>
</tr>
<tr>
<td>NULL_COUNT</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>The number of rows in the table that have NULL for the column.</td>
</tr>
<tr>
<td>NULL_FRACTION</td>
<td>REAL</td>
<td>23</td>
<td>YES</td>
<td>The ratio of NULL records to all records:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NULL_COUNT / TOTAL_ROW_COUNT</td>
</tr>
<tr>
<td>MIN_VALUE</td>
<td>VARCHAR</td>
<td>32672</td>
<td>YES</td>
<td>The minimum value for the column.</td>
</tr>
<tr>
<td>MAX_VALUE</td>
<td>VARCHAR</td>
<td>32672</td>
<td>YES</td>
<td>The maximum value for the column.</td>
</tr>
<tr>
<td>QUANTILES</td>
<td>VARCHAR</td>
<td>32672</td>
<td>YES</td>
<td>The quantiles statistics sketch for the column.</td>
</tr>
<tr>
<td>FREQUENCIES</td>
<td>VARCHAR</td>
<td>32672</td>
<td>YES</td>
<td>The frequencies statistics sketch for the column.</td>
</tr>
<tr>
<td>THETA</td>
<td>VARCHAR</td>
<td>32672</td>
<td>YES</td>
<td>The theta statistics sketch for the column.</td>
</tr>
</tbody>
</table>

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/](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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMAID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Schema ID for the conglomerate</td>
</tr>
<tr>
<td>TABLEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Identifier for table (join with SYSTABLES.TABLEID)</td>
</tr>
<tr>
<td>CONGLOMERATENUMBER</td>
<td>BIGINT</td>
<td>19</td>
<td>NO</td>
<td>Conglomerate ID for the conglomerate (heap or index)</td>
</tr>
<tr>
<td>CONGLOMERATENAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>YES</td>
<td>Index name, if conglomerate is an index, otherwise the table ID</td>
</tr>
<tr>
<td>ISINDEX</td>
<td>BOOLEAN</td>
<td>1</td>
<td>NO</td>
<td>Whether or not conglomerate is an index</td>
</tr>
<tr>
<td>DESCRIPTOR</td>
<td>org.apache.splicemachine.catalog.IndexDescriptor</td>
<td>-1</td>
<td>YES</td>
<td>System type describing the index</td>
</tr>
<tr>
<td>ISCONSTRAINT</td>
<td>BOOLEAN</td>
<td>1</td>
<td>YES</td>
<td>Whether or not the conglomerate is a system-generated index enforcing a constraint</td>
</tr>
<tr>
<td>CONGLOMERATEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for the conglomerate</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINTID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for constraint</td>
</tr>
<tr>
<td>TABLEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Identifier for table (join with SYSTABLES.TABLEID)</td>
</tr>
<tr>
<td>CONSTRAINTNAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>Constraint name (internally generated if not specified by user)</td>
</tr>
<tr>
<td>TYPE</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>›› 'P' for primary key)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>›› 'U' for unique)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>›› 'C' for check)</td>
</tr>
<tr>
<td>SCHEMAID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Identifier for schema that the constraint belongs to (join with SYSSCHEMAS.SCHEMAID)</td>
</tr>
<tr>
<td>STATE</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>›› 'E' for enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>›› 'D' for disabled</td>
</tr>
<tr>
<td>REFERENCECOUNT</td>
<td>INTEGER</td>
<td>10</td>
<td>NO</td>
<td>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</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPENDENTID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>A unique identifier for the dependent</td>
</tr>
<tr>
<td>DEPENDENTFINDER</td>
<td>com.splicemachine.db.catalog.TypeDescriptor</td>
<td>-1</td>
<td>NO</td>
<td>A system type that describes the view, constraint, or trigger that is the dependent</td>
</tr>
<tr>
<td>PROVIDERID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>A unique identifier for the provider</td>
</tr>
<tr>
<td>PROVIDERFINDER</td>
<td>com.splicemachine.db.catalog.TypeDescriptor</td>
<td>-1</td>
<td>NO</td>
<td>A system type that describes the table, conglomerate, constraint, and privilege that is the provider</td>
</tr>
</tbody>
</table>
SYSFILES System Table

The SYSFILES table describes jar files stored in the database.

The following table shows the contents of the SYSFILES system table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for the jar file</td>
</tr>
<tr>
<td>SCHEMAID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>ID of the jar file's schema (join with SYSSCHEMAS.SCHEMAID)</td>
</tr>
<tr>
<td>FILENAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>SQL name of the jar file</td>
</tr>
<tr>
<td>GENERATIONID</td>
<td>BIGINT</td>
<td>19</td>
<td>NO</td>
<td>Generation number for the file. When jar files are replaced, their generation identifiers are changed.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINTID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for the foreign key constraint (join with SYS_CONSTRAINTS.CONSTRAINTID)</td>
</tr>
<tr>
<td>CONGLOMERATEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for index backing up the foreign key constraint (join with SYS_CONGLOMERATES.CONGLOMERATEID)</td>
</tr>
<tr>
<td>KEYCONSTRAINTID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for the primary key or unique constraint referenced by this foreign key (SYSKEYS.CONSTRAINTID or SYS_CONSTRAINTS.CONSTRAINTID)</td>
</tr>
<tr>
<td>DELETERULE</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'R' for NO ACTION (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'S' for RESTRICT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'C' for CASCADE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'U' for SET NULL</td>
</tr>
<tr>
<td>UPDATERULE</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'R' for NO ACTION (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'S' for RESTRICT</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINTID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for constraint</td>
</tr>
<tr>
<td>CONGLOMERATEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for backing index</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>UUID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>The unique ID of the permission. This is the primary key.</td>
</tr>
<tr>
<td>OBJECTTYPE</td>
<td>VARCHAR</td>
<td>36</td>
<td>NO</td>
<td>The kind of object receiving the permission. The only valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'SEQUENCE'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'TYPE'</td>
</tr>
<tr>
<td>OBJECTID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>The UUID of the object receiving the permission.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For sequence generators, the only valid values are SEQUENCEIDs in the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SYS.SYSSEQUENCES table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For user-defined types, the only valid values are ALIASIDs in the SYS.SYS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SYALIASES table if the SYALIASES row describes a user-defined type.</td>
</tr>
<tr>
<td>PERMISSION</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>The type of the permission. The only valid value is 'USAGE'.</td>
</tr>
<tr>
<td>GRANTOR</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The authorization ID of the user who granted the privilege.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Privileges can be granted only by the object owner.</td>
</tr>
<tr>
<td>GRANTEE</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The authorization ID of the user or role to which the privilege was</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>granted</td>
</tr>
<tr>
<td>ISGRANTABLE</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>If the GRANTEE is the owner of the sequence generator or user-defined</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>type, this value is 'Y'.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If the GRANTEE is not the owner of the sequence generator or user-defined</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>type, this value is 'N'.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>UUID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>A unique identifier for this role</td>
</tr>
<tr>
<td>ROLEID</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The role name, after conversion to case normal form</td>
</tr>
<tr>
<td>GRANTEE</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>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.</td>
</tr>
<tr>
<td>GRANTOR</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>This is the authorization identifier of the user that granted this role. If the row represents a role definition, this is the authorization identifier _SYSTEM. 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).</td>
</tr>
<tr>
<td>WITHADMINOPTION</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>A role definition is modelled as a grant from _SYSTEM to the database owner, so if the row represents a role definition, the value is always 'Y'. This means that the creator (the database owner) is always allowed to grant the newly created role. Currently roles cannot be granted WITH ADMIN OPTION, so if the row represents a role grant, the value is always 'N'.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Length</td>
<td>Nullable</td>
<td>Contents</td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>--------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ISDEF</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>If the row represents a role definition, this value is 'Y'. If the row represents a role grant, the value is 'N'.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUTINEPERMSID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Used by the dependency manager to track the dependency of a view, trigger, or constraint on the routine level permissions</td>
</tr>
<tr>
<td>GRANTEE</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The authorization ID of the user or role to which the privilege is granted</td>
</tr>
<tr>
<td>GRANTOR</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The authorization ID of the user who granted the privilege. Privileges can be granted only by the object owner.</td>
</tr>
<tr>
<td>ALIASID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>The ID of the object of the required permission. If PERMTYPE='E', the ALIASID is a reference to the SYS.SYSALIASES table. Otherwise, the ALIASID is a reference to the SYS.SYSTABLES table.</td>
</tr>
<tr>
<td>GRANTOPTION</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Specifies if the GRANTEE is the owner of the routine. Valid values are 'Y' and 'N'.</td>
</tr>
</tbody>
</table>
SYSSCHEMAS System Table

The SYSSCHEMAS table describes the schemas within the current database.

The following table shows the contents of the SYSSCHEMAS system table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMAID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for the schema</td>
</tr>
<tr>
<td>SCHEMANAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>Schema name</td>
</tr>
<tr>
<td>AUTHORIZATIONID</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The authorization identifier of the owner of the schema</td>
</tr>
</tbody>
</table>

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 `SYSCS_UTIL.SYSCS_PEEK_AT_SEQUENCE` system function.

The following table shows the contents of the SYSSEQUENCES system table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQUENCEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>The ID of the sequence generator. This is the primary key.</td>
</tr>
<tr>
<td>SEQUENCENAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The name of the sequence generator. There is a unique index on <code>SEQUENCENAME</code>.</td>
</tr>
<tr>
<td>SCHEMAID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>The ID of the schema that holds the sequence generator. There is a foreign key linking this column to <code>SYSSCHEMAS.SCHEMAID</code>.</td>
</tr>
<tr>
<td>SEQUENCEDATATYPE</td>
<td><code>com.splicemachine.db.catalog.TypeDescriptor</code></td>
<td>-1</td>
<td>NO</td>
<td>System type that describes the precision, length, scale, nullability, type name, and storage type of the data.</td>
</tr>
<tr>
<td>CURRENTVALUE</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>The current value of the sequence generator. This is not the actual next value for the sequence generator. That value can be obtained by calling <code>SYSCS_UTIL.SYSCS_PEEK_AT_SEQUENCE</code>. The initial value is <code>STARTVALUE</code>. This column is NULL when the generator is exhausted and cannot issue any more numbers.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Length</td>
<td>Nullable</td>
<td>Contents</td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
<td>--------</td>
<td>----------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>STARTVALUE</td>
<td>BIGINT</td>
<td>19</td>
<td>NO</td>
<td>The initial value of the sequence generator</td>
</tr>
<tr>
<td>MINIMUMVALUE</td>
<td>BIGINT</td>
<td>19</td>
<td>NO</td>
<td>The minimum value of the sequence generator</td>
</tr>
<tr>
<td>MAXIMUMVALUE</td>
<td>BIGINT</td>
<td>19</td>
<td>NO</td>
<td>The maximum value of the sequence generator</td>
</tr>
<tr>
<td>INCREMENT</td>
<td>BIGINT</td>
<td>19</td>
<td>NO</td>
<td>The step size of the sequence generator</td>
</tr>
<tr>
<td>CYCLEOPTION</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>If the sequence generator cycles, this value is 'Y'. If the sequence generator does not cycle, this value is 'N'.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNAPSHOTNAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The name of the snapshot</td>
</tr>
<tr>
<td>SCHEMANAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>Schema name</td>
</tr>
<tr>
<td>OBJECTNAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The name of the table or index</td>
</tr>
<tr>
<td>CONGLOMERATENUMBER</td>
<td>BIGINT</td>
<td>19</td>
<td>NO</td>
<td>The conglomerate number of the object</td>
</tr>
<tr>
<td>CREATIONTIME</td>
<td>TIMESTAMP</td>
<td>29</td>
<td>NO</td>
<td>The time at which the snapshot was taken</td>
</tr>
<tr>
<td>LASTRESTORETIME</td>
<td>TIMESTAMP</td>
<td>29</td>
<td>NO</td>
<td>The time at which the snapshot was most recently restored</td>
</tr>
</tbody>
</table>

---

**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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>STMTID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for the statement</td>
</tr>
<tr>
<td>STMTNAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>Name of the statement</td>
</tr>
<tr>
<td>SCHEMAID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>The schema in which the statement resides</td>
</tr>
<tr>
<td>TYPE</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Always 'S'</td>
</tr>
<tr>
<td>VALID</td>
<td>BOOLEAN</td>
<td>1</td>
<td>NO</td>
<td>Whether or not the statement is valid</td>
</tr>
<tr>
<td>TEXT</td>
<td>LONG VARCHAR</td>
<td>32,700</td>
<td>NO</td>
<td>Text of the statement</td>
</tr>
<tr>
<td>LASTCOMPILED</td>
<td>TIMESTAMP</td>
<td>29</td>
<td>YES</td>
<td>Time that the statement was compiled</td>
</tr>
<tr>
<td>COMPIILATIONSCHEMAID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>ID of the schema containing the statement</td>
</tr>
<tr>
<td>USINGTEXT</td>
<td>LONG VARCHAR</td>
<td>32,700</td>
<td>YES</td>
<td>Text of the USING clause of the CREATE STATEMENT and ALTER STATEMENT statements</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLEPERMSID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Used by the dependency manager to track the dependency of a view, trigger, or constraint on the table level permissions</td>
</tr>
<tr>
<td>GRANTEE</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The authorization ID of the user or role to which the privilege is granted</td>
</tr>
<tr>
<td>GRANTOR</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The authorization ID of the user who granted the privilege. Privileges can be granted only by the object owner</td>
</tr>
<tr>
<td>TABLEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>The unique identifier for the table on which the permissions have been granted</td>
</tr>
<tr>
<td>SELECTPRIV</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Specifies if the SELECT permission is granted. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; 'y' (non-grantable privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; 'Y' (grantable privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; 'N' (no privilege)</td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Length</td>
<td>Nullable</td>
<td>Contents</td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
<td>--------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>DELETEPRIV</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Specifies if the <strong>DELETE</strong> permission is granted. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'y' (non-grantable privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'Y' (grantable privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'N' (no privilege)</td>
</tr>
<tr>
<td>INSERTPRIV</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Specifies if the <strong>INSERT</strong> permission is granted. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'y' (non-grantable privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'Y' (grantable privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'N' (no privilege)</td>
</tr>
<tr>
<td>UPDATEPRIV</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Specifies if the <strong>UPDATE</strong> permission is granted. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'y' (non-grantable privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'Y' (grantable privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'N' (no privilege)</td>
</tr>
<tr>
<td>REFERENCESPRIV</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Specifies if the <strong>REFERENCE</strong> permission is granted. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'y' (non-grantable privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'Y' (grantable privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'N' (no privilege)</td>
</tr>
<tr>
<td>TRIGGERPRIV</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Specifies if the <strong>TRIGGER</strong> permission is granted. The valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'y' (non-grantable privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'Y' (grantable privilege)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓ 'N' (no privilege)</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for table or view</td>
</tr>
<tr>
<td>TABLENAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>Table or view name</td>
</tr>
<tr>
<td>TABLETYPE</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; 'S' (system table)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; 'T' (user table)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; 'A' (synonym)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; 'V' (view)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; 'E' (external table)</td>
</tr>
<tr>
<td>SCHEMAID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Schema ID for the table or view</td>
</tr>
<tr>
<td>LOCKGRANULARITY</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Lock granularity for the table:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; 'T' (table level locking)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt; 'R' (row level locking, the default)</td>
</tr>
<tr>
<td>VERSION</td>
<td>VARCHAR</td>
<td>128</td>
<td>YES</td>
<td>Version ID.</td>
</tr>
</tbody>
</table>

**See Also**

- About System Tables
SYSTABLESTATISTICS System Table View

The SYSTABLESTATISTICS system 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 view.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMANAME</td>
<td>VARCHAR</td>
<td>32672</td>
<td>YES</td>
<td>The name of the schema</td>
</tr>
<tr>
<td>TABLENAME</td>
<td>VARCHAR</td>
<td>32672</td>
<td>YES</td>
<td>The name of the table</td>
</tr>
<tr>
<td>CONGLOMERATENAME</td>
<td>VARCHAR</td>
<td>32672</td>
<td>YES</td>
<td>The name of the table</td>
</tr>
<tr>
<td>TOTAL_ROW_COUNT</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>The total number of rows in the table</td>
</tr>
<tr>
<td>AVG_ROW_COUNT</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>The average number of rows in the table</td>
</tr>
<tr>
<td>TOTAL_SIZE</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>The total size of the table</td>
</tr>
<tr>
<td>NUM_PARTITIONS</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>The number of partitions(^1) for which statistics were collected.</td>
</tr>
<tr>
<td>AVG_PARTITION_SIZE</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>The average size of a single partition(^1), in bytes.</td>
</tr>
<tr>
<td>ROW_WIDTH</td>
<td>BIGINT</td>
<td>19</td>
<td>YES</td>
<td>The maximum average of the widths of rows in the table, across all partitions, in bytes. Each partition records the average width of a single row. This value is the maximum of those averages across all partitions.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Length</td>
<td>Nullable</td>
<td>Contents</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>--------</td>
<td>----------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STATS_TYPE</td>
<td>INTEGER</td>
<td>10</td>
<td>YES</td>
<td>The type of statistics, which is one of these values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If this value is NULL, 0 is used.</td>
</tr>
<tr>
<td>SAMPLE_FRACTION</td>
<td>DOUBLE</td>
<td>52</td>
<td>YES</td>
<td>The sampling percentage, expressed as 0.0 to 1.0,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>» If statsType=0 (full statistics), this value is not used, and is shown as 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>» If statsType=1, 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).</td>
</tr>
</tbody>
</table>

1 Currently, 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 view
The SYSTRIGGERS table describes the database’s triggers.

The following table shows the contents of the SYSTRIGGERS system table.

### SYSTRIGGERS System Table

The SYSTRIGGERS table describes the database’s triggers.

The following table shows the contents of the SYSTRIGGERS system table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIGGERID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for the trigger</td>
</tr>
<tr>
<td>TRIGGERNAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>Name of the trigger</td>
</tr>
<tr>
<td>SCHEMAID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>ID of the trigger's schema (join with SYSSCHEMAS.SCHEMAID)</td>
</tr>
<tr>
<td>CREATIONTIMESTAMP</td>
<td>TIMESTAMP</td>
<td>29</td>
<td>NO</td>
<td>Time the trigger was created</td>
</tr>
<tr>
<td>EVENT</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>’U’ for update</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>’D’ for delete</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>’I’ for insert</td>
</tr>
<tr>
<td>FIRINGTIME</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>’B’ for before</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>’A’ for after</td>
</tr>
<tr>
<td>TYPE</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>’R’ for row</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>’S’ for statement</td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Length</td>
<td>Nullable</td>
<td>Contents</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------</td>
<td>---------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STATE</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>» 'E' for enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>» 'D' for disabled</td>
</tr>
<tr>
<td>TABLEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>ID of the table on which the trigger is defined</td>
</tr>
<tr>
<td>WHENSTMTID</td>
<td>CHAR</td>
<td>36</td>
<td>YES</td>
<td>Used only if there is a WHEN clause (not yet supported)</td>
</tr>
<tr>
<td>ACTIONSTMTID</td>
<td>CHAR</td>
<td>36</td>
<td>YES</td>
<td>ID of the stored prepared statement for the triggered-SQL-statement (join with SYSSTATEMENTS.STMTID)</td>
</tr>
<tr>
<td>REFERENCEDCOLUMNS</td>
<td>org.apache.Splice Machine.catalog.ReferencedColumns This class is not part of the public API.</td>
<td>-1</td>
<td>YES</td>
<td>Descriptor of the columns to be updated, if this trigger is an update trigger (that is, if the EVENT column contains 'U')</td>
</tr>
<tr>
<td>TRIGGERDEFINITION</td>
<td>LONG VARCHAR</td>
<td>2,147,483,647</td>
<td>YES</td>
<td>Text of the action SQL statement</td>
</tr>
<tr>
<td>REFERENCINGOLD</td>
<td>BOOLEAN</td>
<td>1</td>
<td>YES</td>
<td>Whether or not the OLDREFERENCINGNAME, if non-null, refers to the OLD row or table</td>
</tr>
<tr>
<td>REFERENCINGNEW</td>
<td>BOOLEAN</td>
<td>1</td>
<td>YES</td>
<td>Whether or not the NEWREFERENCINGNAME, if non-null, refers to the NEW row or table</td>
</tr>
<tr>
<td>OLDREFERENCINGNAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>YES</td>
<td>Pseudoname as set using the REFERENCING OLD AS clause</td>
</tr>
<tr>
<td>NEWREFERENCINGNAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>YES</td>
<td>Pseudoname as set using the REFERENCING NEW AS clause</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERNAME</td>
<td>VARCHAR</td>
<td>128</td>
<td>NO</td>
<td>The user’s name, the value of the <code>user</code> attribute on a connection URL.</td>
</tr>
<tr>
<td>HASHINGSHEME</td>
<td>VARCHAR</td>
<td>32672</td>
<td>NO</td>
<td>Describes how the password is hashed.</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>VARCHAR</td>
<td>32672</td>
<td>NO</td>
<td>The password after applying the <code>HASHINGSHEME</code>.</td>
</tr>
<tr>
<td>LASTMODIFIED</td>
<td>TIMESTAMP</td>
<td>29</td>
<td>NO</td>
<td>The time when the password was last updated.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Length</th>
<th>Nullable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLEID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>Unique identifier for the view (join with SYSTABLES.TABLEID)</td>
</tr>
<tr>
<td>VIEWDEFINITION</td>
<td>LONG VARCHAR</td>
<td>32,700</td>
<td>NO</td>
<td>Text of view definition</td>
</tr>
<tr>
<td>CHECKOPTION</td>
<td>CHAR</td>
<td>1</td>
<td>NO</td>
<td>'N' (check option not supported yet)</td>
</tr>
<tr>
<td>COMPILATIONSHEMAID</td>
<td>CHAR</td>
<td>36</td>
<td>NO</td>
<td>ID of the schema containing the view</td>
</tr>
</tbody>
</table>

**See Also**

» About System Tables
## Error Codes

This section contains descriptions of Splice Machine error codes, in these topics:

<table>
<thead>
<tr>
<th>Error Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Warning Messages</td>
</tr>
<tr>
<td>07</td>
<td>Dynamic SQL Error</td>
</tr>
<tr>
<td>08</td>
<td>Connection Exception</td>
</tr>
<tr>
<td>0A</td>
<td>Feature not supported</td>
</tr>
<tr>
<td>0P</td>
<td>Invalid role specification</td>
</tr>
<tr>
<td>21</td>
<td>Cardinality Violation</td>
</tr>
<tr>
<td>22</td>
<td>Data Exception</td>
</tr>
<tr>
<td>23</td>
<td>Constraint Violation</td>
</tr>
<tr>
<td>24</td>
<td>Invalid Cursor State</td>
</tr>
<tr>
<td>25</td>
<td>Invalid Transaction State</td>
</tr>
<tr>
<td>28</td>
<td>Invalid Authorization Specification</td>
</tr>
<tr>
<td>2D</td>
<td>Invalid Transaction Termination</td>
</tr>
<tr>
<td>38</td>
<td>External Function Exception</td>
</tr>
<tr>
<td>39</td>
<td>External Routine Invocation Exception</td>
</tr>
<tr>
<td>3B</td>
<td>Invalid SAVEPOINT</td>
</tr>
<tr>
<td>40</td>
<td>Transaction Rollback</td>
</tr>
<tr>
<td>42</td>
<td>Syntax Error or Access Rule Violation</td>
</tr>
<tr>
<td>57</td>
<td>DRDA Network Protocol - Execution Failure</td>
</tr>
<tr>
<td>58</td>
<td>DRDA Network Protocol - Protocol Error</td>
</tr>
<tr>
<td>X0</td>
<td>Execution exceptions</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>XBCA</td>
<td>CacheService</td>
</tr>
<tr>
<td>XBCM</td>
<td>ClassManager</td>
</tr>
<tr>
<td>XBCX</td>
<td>Cryptography</td>
</tr>
<tr>
<td>XBM</td>
<td>Monitor</td>
</tr>
<tr>
<td>XCL</td>
<td>Execution exceptions</td>
</tr>
<tr>
<td>XCW</td>
<td>Upgrade unsupported</td>
</tr>
<tr>
<td>XCX</td>
<td>Internal Utility Errors</td>
</tr>
<tr>
<td>XCY</td>
<td>Derby Property Exceptions</td>
</tr>
<tr>
<td>XCZ</td>
<td>org.apache.derby.database.UserUtility</td>
</tr>
<tr>
<td>XD00</td>
<td>Dependency Manager</td>
</tr>
<tr>
<td>XIE</td>
<td>Import/Export Exceptions</td>
</tr>
<tr>
<td>XJ</td>
<td>Connectivity Errors</td>
</tr>
<tr>
<td>XK</td>
<td>Security Exceptions</td>
</tr>
<tr>
<td>XN</td>
<td>Network Client Exceptions</td>
</tr>
<tr>
<td>XRE</td>
<td>Replication Exceptions</td>
</tr>
<tr>
<td>XSAI</td>
<td>Store - access.protocol.interface</td>
</tr>
<tr>
<td>XSAM</td>
<td>Store - AccessManager</td>
</tr>
<tr>
<td>XSAS</td>
<td>Store - Sort</td>
</tr>
<tr>
<td>XSAX</td>
<td>Store - access.protocol.XA statement</td>
</tr>
<tr>
<td>XSCB</td>
<td>Store - BTree</td>
</tr>
<tr>
<td>XSCG0</td>
<td>Conglomerate</td>
</tr>
<tr>
<td>XSCH</td>
<td>Heap</td>
</tr>
<tr>
<td>XSDA</td>
<td>RawStore - Data.Generic statement</td>
</tr>
<tr>
<td>XSDB</td>
<td>RawStore - Data.Generic transaction</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>XSDF</td>
<td>RawStore - Data.Filesystem statement</td>
</tr>
<tr>
<td>XSDG</td>
<td>RawStore - Data.Filesystem database</td>
</tr>
<tr>
<td>XSLA</td>
<td>RawStore - Log.Generic database exceptions</td>
</tr>
<tr>
<td>XSLB</td>
<td>RawStore - Log.Generic statement exceptions</td>
</tr>
<tr>
<td>XSRS</td>
<td>RawStore - protocol.Interface statement</td>
</tr>
<tr>
<td>XSTA2</td>
<td>XACT_TRANSACTION_ACTIVE</td>
</tr>
<tr>
<td>XSTB</td>
<td>RawStore - Transactions.Basic system</td>
</tr>
<tr>
<td>XXXXX</td>
<td>No SQLSTATE</td>
</tr>
</tbody>
</table>
## Error Class 01: Warning Messages

### Error Class 01: Warnings

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>01001</td>
<td>An attempt to update or delete an already deleted row was made: No row was updated or deleted.</td>
</tr>
<tr>
<td>01003</td>
<td>Null values were eliminated from the argument of a column function.</td>
</tr>
<tr>
<td>01006</td>
<td>Privilege not revoked from user <code>&lt;authorizationID&gt;</code>.</td>
</tr>
<tr>
<td>01007</td>
<td>Role <code>&lt;authorizationID&gt;</code> not revoked from authentication id <code>&lt;authorizationID&gt;</code>.</td>
</tr>
<tr>
<td>01008</td>
<td>WITH ADMIN OPTION of role <code>&lt;authorizationID&gt;</code> not revoked from authentication id <code>&lt;authorizationID&gt;</code>.</td>
</tr>
<tr>
<td>01009</td>
<td>Generated column <code>&lt;columnName&gt;</code> dropped from table <code>&lt;tableName&gt;</code>.</td>
</tr>
<tr>
<td>0100E</td>
<td>XX Attempt to return too many result sets.</td>
</tr>
<tr>
<td>01500</td>
<td>The constraint <code>&lt;constraintName&gt;</code> on table <code>&lt;tableName&gt;</code> has been dropped.</td>
</tr>
<tr>
<td>01501</td>
<td>The view <code>&lt;viewName&gt;</code> has been dropped.</td>
</tr>
<tr>
<td>01502</td>
<td>The trigger <code>&lt;triggerName&gt;</code> on table <code>&lt;tableName&gt;</code> has been dropped.</td>
</tr>
<tr>
<td>01503</td>
<td>The column <code>&lt;columnName&gt;</code> on table <code>&lt;tableName&gt;</code> has been modified by adding a not null constraint.</td>
</tr>
<tr>
<td>01504</td>
<td>The new index is a duplicate of an existing index: <code>&lt;indexName&gt;</code>.</td>
</tr>
<tr>
<td>01505</td>
<td>The value <code>&lt;valueName&gt;</code> may be truncated.</td>
</tr>
<tr>
<td>01522</td>
<td>The newly defined synonym ' <code>&lt;synonymName&gt;' resolved to the object ' </code>&lt;objectName&gt;' which is currently undefined.</td>
</tr>
<tr>
<td>01J01</td>
<td>Database ' `&lt;databaseName&gt;' not created, connection made to existing database instead.</td>
</tr>
<tr>
<td>01J02</td>
<td>Scroll sensitive cursors are not currently implemented.</td>
</tr>
<tr>
<td>01J04</td>
<td>The class ' <code>&lt;className&gt;' for column ' </code>&lt;columnName&gt;' does not implement java.io.Serializable or java.sql.SQLData. Instances must implement one of these interfaces to allow them to be stored.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>01J05</td>
<td>Database upgrade succeeded. The upgraded database is now ready for use. Revalidating stored prepared statements failed. See next exception for details of failure.</td>
</tr>
<tr>
<td>01J06</td>
<td>ResultSet not updatable. Query does not qualify to generate an updatable ResultSet.</td>
</tr>
<tr>
<td>01J07</td>
<td>ResultSetHoldability restricted to ResultSet.CLOSE_CURSORS_AT_COMMIT for a global transaction.</td>
</tr>
<tr>
<td>01J08</td>
<td>Unable to open resultSet type &lt;resultSetType&gt;. ResultSet type &lt;resultSetType&gt; opened.</td>
</tr>
<tr>
<td>01J10</td>
<td>Scroll sensitive result sets are not supported by server; remapping to forward-only cursor</td>
</tr>
<tr>
<td>01J12</td>
<td>Unable to obtain message text from server. See the next exception. The stored procedure SYSIBM.SQLCAMEMESSAGE is not installed on the server. Please contact your database administrator.</td>
</tr>
<tr>
<td>01J13</td>
<td>Number of rows returned (&lt;number&gt;) is too large to fit in an integer; the value returned will be truncated.</td>
</tr>
<tr>
<td>01J14</td>
<td>SQL authorization is being used without first enabling authentication.</td>
</tr>
<tr>
<td>01J15</td>
<td>Your password will expire in &lt;remainingDays&gt; day(s). Please use the SYSCS_UTIL.SYSCS_MODIFY_PASSWORD procedure to change your password in database '&lt;databaseName&gt;'.</td>
</tr>
<tr>
<td>01J16</td>
<td>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 '&lt;databaseName&gt;'.</td>
</tr>
<tr>
<td>01J17</td>
<td>Statistics are unavailable or out of date for one or more tables involved in this query.</td>
</tr>
</tbody>
</table>
## Error Class 07: Dynamic SQL Errors

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>07000</td>
<td>At least one parameter to the current statement is uninitialized.</td>
</tr>
<tr>
<td>07004</td>
<td>Parameter <code>&lt;parameterName&gt;</code> is an <code>&lt;procedureName&gt;</code> procedure parameter and must be registered with <code>CallableStatement.registerOutParameter</code> before execution.</td>
</tr>
<tr>
<td>07009</td>
<td>No input parameters.</td>
</tr>
</tbody>
</table>
### Error Class 08: Connection Exception

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>08000</td>
<td>Connection closed by unknown interrupt.</td>
</tr>
<tr>
<td>08001.C.10</td>
<td>A connection could not be established because the security token is larger than the maximum allowed by the network protocol.</td>
</tr>
<tr>
<td>08001.C.11</td>
<td>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.</td>
</tr>
<tr>
<td>08001.C.12</td>
<td>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.</td>
</tr>
<tr>
<td>08001.C.13</td>
<td>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.</td>
</tr>
<tr>
<td>08001.C.14</td>
<td>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.</td>
</tr>
<tr>
<td>08001.C.1</td>
<td>Required Splice DataSource property <code>&lt;propertyName&gt;</code> not set.</td>
</tr>
<tr>
<td>08001.C.2</td>
<td><code>&lt;error&gt;</code>: Error connecting to server <code>&lt;serverName&gt;</code> on port <code>&lt;portNumber&gt;</code> with message <code>&lt;messageText&gt;</code>.</td>
</tr>
<tr>
<td>08001.C.3</td>
<td>SocketException: '&lt;error&gt;'</td>
</tr>
<tr>
<td>08001.C.4</td>
<td>Unable to open stream on socket: '&lt;error&gt;'.</td>
</tr>
<tr>
<td>08001.C.5</td>
<td>User id length (<code>&lt;number&gt;</code>) is outside the range of 1 to <code>&lt;number&gt;</code>.</td>
</tr>
<tr>
<td>08001.C.6</td>
<td>Password length (<code>&lt;value&gt;</code>) is outside the range of 1 to <code>&lt;number&gt;</code>.</td>
</tr>
<tr>
<td>08001.C.7</td>
<td>User id can not be null.</td>
</tr>
<tr>
<td>08001.C.8</td>
<td>Password can not be null.</td>
</tr>
<tr>
<td>08001.C.9</td>
<td>A connection could not be established because the database name <code>'&lt;databaseName&gt;'</code> is larger than the maximum length allowed by the network protocol.</td>
</tr>
<tr>
<td>08003</td>
<td>No current connection.</td>
</tr>
<tr>
<td>08003.C.1</td>
<td>getConnection() is not valid on a closed PooledConnection.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>08003.C.2</td>
<td>Lob method called after connection was closed</td>
</tr>
<tr>
<td>08003.C.3</td>
<td>The underlying physical connection is stale or closed.</td>
</tr>
<tr>
<td>08004</td>
<td>Connection refused: <code>&lt;connectionName&gt;</code></td>
</tr>
<tr>
<td>08004.C.1</td>
<td>Connection authentication failure occurred. Reason: <code>&lt;reasonText&gt;</code>.</td>
</tr>
<tr>
<td>08004.C.2</td>
<td>The connection was refused because the database <code>&lt;databaseName&gt;</code> was not found.</td>
</tr>
<tr>
<td>08004.C.3</td>
<td>Database connection refused.</td>
</tr>
<tr>
<td>08004.C.4</td>
<td>User ‘&lt;authorizationID&gt;’ cannot shut down database ‘&lt;databaseName&gt;’. Only the database owner can perform this operation.</td>
</tr>
<tr>
<td>08004.C.5</td>
<td>User ‘&lt;authorizationID&gt;’ cannot (re)encrypt database ‘&lt;databaseName&gt;’. Only the database owner can perform this operation.</td>
</tr>
<tr>
<td>08004.C.6</td>
<td>User ‘&lt;authorizationID&gt;’ cannot hard upgrade database ‘&lt;databaseName&gt;’. Only the database owner can perform this operation.</td>
</tr>
<tr>
<td>08004.C.7</td>
<td>Connection refused to database <code>&lt;databaseName&gt;</code> because it is in replication slave mode.</td>
</tr>
<tr>
<td>08004.C.8</td>
<td>User ‘&lt;authorizationID&gt;’ cannot issue a replication operation on database ‘&lt;databaseName&gt;’. Only the database owner can perform this operation.</td>
</tr>
<tr>
<td>08004.C.9</td>
<td>Missing permission for user ‘&lt;authorizationID&gt;’ to shutdown system <code>[exceptionMsg]</code>.</td>
</tr>
<tr>
<td>08004.C.10</td>
<td>Cannot check system permission to create database ‘&lt;databaseName&gt;’ <code>[exceptionMsg]</code>.</td>
</tr>
<tr>
<td>08004.C.11</td>
<td>Missing permission for user ‘&lt;authorizationID&gt;’ to create database ‘&lt;databaseName&gt;’ <code>[exceptionMsg]</code>.</td>
</tr>
<tr>
<td>08004.C.12</td>
<td>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.</td>
</tr>
<tr>
<td>08004.C.13</td>
<td>Username or password is null or 0 length.</td>
</tr>
<tr>
<td>08006.C</td>
<td>A network protocol error was encountered and the connection has been terminated: <code>&lt;error&gt;</code></td>
</tr>
<tr>
<td>08006.C.1</td>
<td>An error occurred during connect reset and the connection has been terminated. See chained exceptions for details.</td>
</tr>
<tr>
<td>08006.C.2</td>
<td>SocketException: ‘&lt;error&gt;’</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>08006.C.3</td>
<td>A communications error has been detected: &lt;error&gt;.</td>
</tr>
<tr>
<td>08006.C.4</td>
<td>An error occurred during a deferred connect reset and the connection has been terminated. See chained exceptions for details.</td>
</tr>
<tr>
<td>08006.C.5</td>
<td>Insufficient data while reading from the network - expected a minimum of &lt;number&gt; bytes and received only &lt;number&gt; bytes. The connection has been terminated.</td>
</tr>
<tr>
<td>08006.C.6</td>
<td>Attempt to fully materialize lob data that is too large for the JVM. The connection has been terminated.</td>
</tr>
<tr>
<td>08006.C.8</td>
<td>com.splicemachine.db.jdbc.EmbeddedDriver is not registered with the JDBC driver manager</td>
</tr>
<tr>
<td>08006.C.9</td>
<td>Can't execute statement while in Restore Mode. Reboot database after restore operation is finished.</td>
</tr>
<tr>
<td>08006.D</td>
<td>Database '&lt;databaseName&gt;' shutdown.</td>
</tr>
<tr>
<td>08006.D.1</td>
<td>Database '&lt;databaseName&gt;' dropped.</td>
</tr>
</tbody>
</table>
Error Class 0A: Feature Not Supported

### Error Class 0A: Feature Not Supported

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0A000.S</td>
<td>Feature not implemented: <code>&lt;featureName&gt;</code></td>
</tr>
<tr>
<td>0A000.SP</td>
<td>Feature not yet implemented in Splice Machine, but available soon: <code>&lt;featureName&gt;</code></td>
</tr>
<tr>
<td>0A000.C.6</td>
<td>The DRDA command <code>&lt;commandName&gt;</code> is not currently implemented. The connection has been terminated.</td>
</tr>
<tr>
<td>0A000.S.1</td>
<td>JDBC method is not yet implemented.</td>
</tr>
<tr>
<td>0A000.S.2</td>
<td>JDBC method <code>&lt;methodName&gt;</code> is not supported by the server. Please upgrade the server.</td>
</tr>
<tr>
<td>0A000.S.3</td>
<td>resultSetHoldability property <code>&lt;propertyName&gt;</code> not supported</td>
</tr>
<tr>
<td>0A000.S.4</td>
<td>cancel() not supported by the server.</td>
</tr>
<tr>
<td>0A000.S.5</td>
<td>Security mechanism <code>&lt;mechanismName&gt;</code> is not supported.</td>
</tr>
<tr>
<td>0A000.S.7</td>
<td>The data type <code>&lt;datatypeName&gt;</code> is not supported.</td>
</tr>
</tbody>
</table>
## Error Class 0P: Invalid Role Specification

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0P000</td>
<td>Invalid role specification, role does not exist: '&lt;roleName&gt;'.</td>
</tr>
<tr>
<td>0P000.S.1</td>
<td>Invalid role specification, role not granted to current user or PUBLIC: '&lt;roleName&gt;'.</td>
</tr>
</tbody>
</table>
## Error Class 21: Cardinality Violation

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>21000</td>
<td>Scalar subquery is only allowed to return a single row.</td>
</tr>
</tbody>
</table>
Error Class 22: Data Exception

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>22001</td>
<td>A truncation error was encountered trying to shrink <code>&lt;value&gt;</code> '&lt;value&gt;' to length <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>22003</td>
<td>The resulting value is outside the range for the data type <code>&lt;datatypeName&gt;</code>.</td>
</tr>
<tr>
<td>22003.S.0</td>
<td>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 <code>&lt;modifiedRowCount&gt;</code>.</td>
</tr>
<tr>
<td>22003.S.1</td>
<td>Year <code>&lt;value&gt;</code> exceeds the maximum <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>22003.S.2</td>
<td>Decimal may only be up to 38 digits.</td>
</tr>
<tr>
<td>22003.S.3</td>
<td>Overflow occurred during numeric data type conversion of <code>&lt;datatypeName&gt;</code> to <code>&lt;datatypeName&gt;</code>.</td>
</tr>
<tr>
<td>22003.S.4</td>
<td>The length <code>&lt;number&gt;</code> exceeds the maximum length <code>&lt;datatypeName&gt;</code> for the data type.</td>
</tr>
<tr>
<td>22005.S.1</td>
<td>Unable to convert a value of type <code>&lt;typeName&gt;</code> to type <code>&lt;typeName&gt;</code>: the encoding is not supported.</td>
</tr>
<tr>
<td>22005.S.2</td>
<td>The required character converter is not available.</td>
</tr>
<tr>
<td>22005.S.3</td>
<td>Unicode string cannot convert to Ebcdic string</td>
</tr>
<tr>
<td>22005.S.4</td>
<td>Unrecognized JDBC type. Type: <code>&lt;typeName&gt;</code>, columnCount: <code>&lt;value&gt;</code>, columnIndex: <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>22005.S.5</td>
<td>Invalid JDBC type for parameter <code>&lt;parameterName&gt;</code>.</td>
</tr>
<tr>
<td>22005.S.6</td>
<td>Unrecognized Java SQL type <code>&lt;datatypeName&gt;</code>.</td>
</tr>
<tr>
<td>22005.S.7</td>
<td>Unicode string cannot convert to UTF-8 string</td>
</tr>
<tr>
<td>22005</td>
<td>An attempt was made to get a data value of type <code>&lt;datatypeName&gt;</code> from a data value of type <code>&lt;datatypeName&gt;</code>.</td>
</tr>
<tr>
<td>22007.S.180</td>
<td>The string representation of a datetime value is out of range.</td>
</tr>
<tr>
<td>22007.S.181</td>
<td>The syntax of the string representation of a datetime value is incorrect.</td>
</tr>
<tr>
<td>22008.S</td>
<td><code>&lt;argument&gt;</code> is an invalid argument to the <code>&lt;functionName&gt;</code> function.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>2200H.S</td>
<td>Sequence generator '&lt;schemaName&gt;.&lt;sequenceName&gt;' does not cycle. No more values can be obtained from this sequence generator.</td>
</tr>
<tr>
<td>2200L</td>
<td>Values assigned to XML columns must be well-formed DOCUMENT nodes.</td>
</tr>
<tr>
<td>2200M</td>
<td>Invalid XML DOCUMENT: &lt;parserError&gt;</td>
</tr>
<tr>
<td>2200V</td>
<td>Invalid context item for &lt;operatorName&gt; operator; context items must be well-formed DOCUMENT nodes.</td>
</tr>
<tr>
<td>2200W</td>
<td>XQuery serialization error: Attempted to serialize one or more top-level Attribute nodes.</td>
</tr>
<tr>
<td>22011</td>
<td>The second or third argument of the SUBSTR function is out of range.</td>
</tr>
<tr>
<td>22011.S.1</td>
<td>The range specified for the substring with offset &lt;operatorName&gt; and len &lt;len&gt; is out of range for the String: &lt;str&gt;.</td>
</tr>
<tr>
<td>22012</td>
<td>Attempt to divide by zero.</td>
</tr>
<tr>
<td>22013</td>
<td>Attempt to take the square root of a negative number, '&lt;value&gt;'.</td>
</tr>
<tr>
<td>22014</td>
<td>The start position for LOCATE is invalid; it must be a positive integer. The index to start the search from is '&lt;startIndex&gt;'. The string to search for is '&lt;searchString&gt;'. The string to search from is '&lt;fromString&gt;'.</td>
</tr>
<tr>
<td>22015</td>
<td>The '&lt;functionName&gt;' function is not allowed on the following set of types. First operand is of type '&lt;typeName&gt;'. Second operand is of type '&lt;typeName&gt;'. Third operand (start position) is of type '&lt;typeName&gt;'.</td>
</tr>
<tr>
<td>22018</td>
<td>Invalid character string format for type &lt;typeName&gt;.</td>
</tr>
<tr>
<td>22019</td>
<td>Invalid escape sequence, '&lt;sequenceName&gt;'. The escape string must be exactly one character. It cannot be a null or more than one character.</td>
</tr>
<tr>
<td>22020</td>
<td>Invalid trim string, '&lt;string&gt;'. The trim string must be exactly one character or NULL. It cannot be more than one character.</td>
</tr>
<tr>
<td>22021</td>
<td>Unknown character encoding '&lt;typeName&gt;'.</td>
</tr>
<tr>
<td>22025</td>
<td>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.</td>
</tr>
<tr>
<td>22027</td>
<td>The built-in TRIM() function only supports a single trim character. The LTRIM() and RTRIM() built-in functions support multiple trim characters.</td>
</tr>
<tr>
<td>22028</td>
<td>The string exceeds the maximum length of &lt;number&gt;.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>22501</td>
<td>An ESCAPE clause of NULL returns undefined results and is not allowed.</td>
</tr>
<tr>
<td>2201X</td>
<td>Invalid row count for OFFSET, must be &gt;= 0.</td>
</tr>
<tr>
<td>2201Y</td>
<td>Invalid LEAD, LAG for OFFSET, must be greater or equal to 0 and less than Integer.MAX_VALUE. Got '&lt;value&gt;'.</td>
</tr>
<tr>
<td>2202A</td>
<td>Missing argument for first(), last() function.</td>
</tr>
<tr>
<td>2202B</td>
<td>Missing argument for lead(), lag() function.</td>
</tr>
<tr>
<td>2202C</td>
<td>&quot;default&quot; argument for lead(), lag() function is not implemented.</td>
</tr>
<tr>
<td>2202D</td>
<td>NULL value for data type &lt;string&gt; not supported.</td>
</tr>
<tr>
<td>2202E</td>
<td>A &lt;string&gt; column cannot be aggregated.</td>
</tr>
<tr>
<td>2201W</td>
<td>Row count for FIRST/NEXT/TOP must be &gt;= 1 and row count for LIMIT must be &gt;= 0.</td>
</tr>
<tr>
<td>2201Z</td>
<td>NULL value not allowed for &lt;string&gt; argument.</td>
</tr>
</tbody>
</table>
## Error Class 23: Constraint Violation

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>23502</td>
<td>Column '&lt;columnName&gt;' cannot accept a NULL value.</td>
</tr>
<tr>
<td>23503</td>
<td>&lt;value&gt; on table '&lt;tableName&gt;' caused a violation of foreign key constraint '&lt;constraintName&gt;' for key '&lt;keyName&gt;'. The statement has been rolled back.</td>
</tr>
<tr>
<td>23505</td>
<td>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 '&lt;value&gt;' defined on '&lt;value&gt;'.</td>
</tr>
<tr>
<td>23513</td>
<td>The check constraint '&lt;constraintName&gt;' was violated while performing an INSERT or UPDATE on table '&lt;tableName&gt;'.</td>
</tr>
</tbody>
</table>
### Error Class 24: Invalid Cursor State

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>24000</td>
<td>Invalid cursor state - no current row.</td>
</tr>
<tr>
<td>24501.S</td>
<td>The identified cursor is not open.</td>
</tr>
</tbody>
</table>
## Error Class 25: Invalid Transaction State

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>25001</td>
<td>Cannot close a connection while a transaction is still active.</td>
</tr>
<tr>
<td>25001.S.1</td>
<td>Invalid transaction state: active SQL transaction.</td>
</tr>
<tr>
<td>25501</td>
<td>Unable to set the connection read-only property in an active transaction.</td>
</tr>
<tr>
<td>25502</td>
<td>An SQL data change is not permitted for a read-only connection, user or database.</td>
</tr>
<tr>
<td>25503</td>
<td>DDL is not permitted for a read-only connection, user or database.</td>
</tr>
<tr>
<td>25505</td>
<td>A read-only user or a user in a read-only database is not permitted to disable read-only mode on a connection.</td>
</tr>
</tbody>
</table>
Error Class 28: Invalid Authorization Specification

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>28502</td>
<td>The user name '&lt;authorizationID&gt;' is not valid.</td>
</tr>
</tbody>
</table>
Error Class 2D: Invalid Transaction Termination

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D521.S.1</td>
<td>setAutoCommit(true) invalid during global transaction.</td>
</tr>
<tr>
<td>2D521.S.2</td>
<td>COMMIT or ROLLBACK invalid for application execution environment.</td>
</tr>
</tbody>
</table>
Error Class 38: External Function Exception

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>38000</td>
<td>The exception '&lt;exception&gt;' was thrown while evaluating an expression.</td>
</tr>
<tr>
<td>38001</td>
<td>The external routine is not allowed to execute SQL statements.</td>
</tr>
<tr>
<td>38002</td>
<td>The routine attempted to modify data, but the routine was not defined as MODIFIES SQL DATA.</td>
</tr>
<tr>
<td>38004</td>
<td>The routine attempted to read data, but the routine was not defined as READS SQL DATA.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>39004</td>
<td>A NULL value cannot be passed to a method which takes a parameter of primitive type <code>&lt;type&gt;</code>.</td>
</tr>
</tbody>
</table>
Error Class 3B: Invalid SAVEPOINT

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>3B001.S</td>
<td>Savepoint &lt;savepointName&gt; does not exist or is not active in the current transaction.</td>
</tr>
<tr>
<td>3B002.S</td>
<td>The maximum number of savepoints has been reached.</td>
</tr>
<tr>
<td>3B501.S</td>
<td>A SAVEPOINT with the passed name already exists in the current transaction.</td>
</tr>
<tr>
<td>3B502.S</td>
<td>A RELEASE or ROLLBACK TO SAVEPOINT was specified, but the savepoint does not exist.</td>
</tr>
</tbody>
</table>
## Error Class 40: Transaction Rollback

### Error Class 40: Transaction Rollback

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>40001</td>
<td>A lock could not be obtained due to a deadlock, cycle of locks and waiters is: &lt;lockCycle&gt;. The selected victim is XID: &lt;transactionID&gt;.</td>
</tr>
<tr>
<td>40XC0</td>
<td>Dead statement. This may be caused by catching a transaction severity error inside this statement.</td>
</tr>
<tr>
<td>40XD0</td>
<td>Container has been closed.</td>
</tr>
<tr>
<td>40XD1</td>
<td>Container was opened in read-only mode.</td>
</tr>
<tr>
<td>40XD2</td>
<td>Container &lt;containerName&gt; cannot be opened; it either has been dropped or does not exist.</td>
</tr>
<tr>
<td>40XL1</td>
<td>A lock could not be obtained within the time requested</td>
</tr>
<tr>
<td>40XL1.T.1</td>
<td>A lock could not be obtained within the time requested. The lockTable dump is: &lt;tableDump&gt;</td>
</tr>
<tr>
<td>40XT0</td>
<td>An internal error was identified by RawStore module.</td>
</tr>
<tr>
<td>40XT1</td>
<td>An exception was thrown during transaction commit.</td>
</tr>
<tr>
<td>40XT2</td>
<td>An exception was thrown during rollback of a SAVEPOINT.</td>
</tr>
<tr>
<td>40XT4</td>
<td>An attempt was made to close a transaction that was still active. The transaction has been aborted.</td>
</tr>
<tr>
<td>40XT5</td>
<td>Exception thrown during an internal transaction.</td>
</tr>
<tr>
<td>40XT6</td>
<td>Database is in quiescent state, cannot activate transaction. Please wait for a moment till it exits the quiescent state.</td>
</tr>
<tr>
<td>40XT7</td>
<td>Operation is not supported in an internal transaction.</td>
</tr>
</tbody>
</table>
## Error Class 42: Syntax Error or Access Rule Violation

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>42000</td>
<td>Syntax error or access rule violation; see additional errors for details.</td>
</tr>
<tr>
<td>42500</td>
<td>User '&lt;authorizationID&gt;' does not have &lt;permissionType&gt; permission on table '&lt;schemaName&gt;'.'&lt;tableName&gt;' .</td>
</tr>
<tr>
<td>42501</td>
<td>User '&lt;authorizationID&gt;' does not have &lt;permissionType&gt; permission on table '&lt;schemaName&gt;'.'&lt;tableName&gt;' for grant.</td>
</tr>
<tr>
<td>42502</td>
<td>User '&lt;authorizationID&gt;' does not have &lt;permissionType&gt; permission on column '&lt;columnName&gt;' of table '&lt;schemaName&gt;'.'&lt;tableName&gt;' .</td>
</tr>
<tr>
<td>42503</td>
<td>User '&lt;authorizationID&gt;' does not have &lt;permissionType&gt; permission on column '&lt;columnName&gt;' of table '&lt;schemaName&gt;'.'&lt;tableName&gt;' for grant.</td>
</tr>
<tr>
<td>42504</td>
<td>User '&lt;authorizationID&gt;' does not have &lt;permissionType&gt; permission on &lt;objectName&gt; '&lt;schemaName&gt;'.'&lt;tableName&gt;' .</td>
</tr>
<tr>
<td>42505</td>
<td>User '&lt;authorizationID&gt;' does not have &lt;permissionType&gt; permission on &lt;objectName&gt; '&lt;schemaName&gt;'.'&lt;tableName&gt;' for grant.</td>
</tr>
<tr>
<td>42506</td>
<td>User '&lt;authorizationID&gt;' is not the owner of &lt;objectName&gt; '&lt;schemaName&gt;'.'&lt;tableName&gt;' .</td>
</tr>
<tr>
<td>42507</td>
<td>User '&lt;authorizationID&gt;' can not perform the operation in schema '&lt;schemaName&gt;' .</td>
</tr>
<tr>
<td>42508</td>
<td>User '&lt;authorizationID&gt;' can not create schema '&lt;schemaName&gt;' . Only database owner could issue this statement.</td>
</tr>
<tr>
<td>42509</td>
<td>Specified grant or revoke operation is not allowed on object '&lt;objectName&gt;' .</td>
</tr>
<tr>
<td>4250A</td>
<td>User '&lt;authorizationID&gt;' does not have &lt;permissionName&gt; permission on object '&lt;schemaName&gt;'.'&lt;objectName&gt;' .</td>
</tr>
<tr>
<td>4250B</td>
<td>Invalid database authorization property '&lt;value&gt;=&lt;value&gt;' .</td>
</tr>
<tr>
<td>4250C</td>
<td>User(s) '&lt;authorizationID&gt;' must not be in both read-only and full-access authorization lists.</td>
</tr>
<tr>
<td>4250D</td>
<td>Repeated user(s) '&lt;authorizationID&gt;' in access list '&lt;listName&gt;' ;</td>
</tr>
<tr>
<td>4250E</td>
<td>Internal Error: invalid &lt;authorizationID&gt; id in statement permission list.</td>
</tr>
<tr>
<td>4251A</td>
<td>Statement &lt;value&gt; can only be issued by database owner.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>4251B</td>
<td>PUBLIC is reserved and cannot be used as a user identifier or role name.</td>
</tr>
<tr>
<td>4251C</td>
<td>Role <code>&lt;authorizationID&gt;</code> cannot be granted to <code>&lt;authorizationID&gt;</code> because this would create a circularity.</td>
</tr>
<tr>
<td>4251D</td>
<td>Only the database owner can perform this operation.</td>
</tr>
<tr>
<td>4251E</td>
<td>No one can view the <code>'&lt;tableName&gt;.'&lt;columnName&gt;'</code> column.</td>
</tr>
<tr>
<td>4251F</td>
<td>You cannot drop the credentials of the database owner.</td>
</tr>
<tr>
<td>4251G</td>
<td>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.</td>
</tr>
<tr>
<td>4251H</td>
<td>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).</td>
</tr>
<tr>
<td>4251I</td>
<td>Authentication cannot be performed because the credentials database <code>'&lt;databaseName&gt;'</code> does not exist.</td>
</tr>
<tr>
<td>4251J</td>
<td>The value for the property <code>'&lt;propertyName&gt;'</code> is formatted badly.</td>
</tr>
<tr>
<td>4251K</td>
<td>The first credentials created must be those of the DBO.</td>
</tr>
<tr>
<td>4251L</td>
<td>The derby.authentication.provider property specifies <code>'&lt;dbName&gt;'</code> as the name of the credentials database. This is not a valid name for a database.</td>
</tr>
<tr>
<td>4251M</td>
<td>User <code>'&lt;authorizationID&gt;'</code> does not have <code>&lt;permissionType&gt;</code> permission to analyze table <code>'&lt;schemaName&gt;.'&lt;tableName&gt;'</code>.</td>
</tr>
<tr>
<td>42601</td>
<td>In an ALTER TABLE statement, the column <code>'&lt;columnName&gt;'</code> has been specified as NOT NULL and either the DEFAULT clause was not specified or was specified as DEFAULT NULL.</td>
</tr>
<tr>
<td>42601.S.372</td>
<td>ALTER TABLE statement cannot add an IDENTITY column to a table.</td>
</tr>
<tr>
<td>42605</td>
<td>The number of arguments for function <code>'&lt;functionName&gt;'</code> is incorrect.</td>
</tr>
<tr>
<td>42606</td>
<td>An invalid hexadecimal constant starting with <code>'&lt;number&gt;'</code> has been detected.</td>
</tr>
<tr>
<td>42610</td>
<td>All the arguments to the COALESCE/VALUE function cannot be parameters. The function needs at least one argument that is not a parameter.</td>
</tr>
<tr>
<td>42611</td>
<td>The length, precision, or scale attribute for column, or type mapping <code>'&lt;value&gt;'</code> is not valid.</td>
</tr>
<tr>
<td>42613</td>
<td>Multiple or conflicting keywords involving the <code>'&lt;clause&gt;'</code> clause are present.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>42621</td>
<td>A check constraint or generated column that is defined with '&lt;value&gt;' is invalid.</td>
</tr>
<tr>
<td>42622</td>
<td>The name '&lt;name&gt;' is too long. The maximum length is '&lt;number&gt;'.</td>
</tr>
<tr>
<td>42734</td>
<td>Name '&lt;name&gt;' specified in context '&lt;context&gt;' is not unique.</td>
</tr>
<tr>
<td>42802</td>
<td>The number of values assigned is not the same as the number of specified or implied columns.</td>
</tr>
<tr>
<td>42803</td>
<td>An expression containing the column '&lt;columnName&gt;' appears in the SELECT list and is not part of a GROUP BY clause.</td>
</tr>
<tr>
<td>42815.S.713</td>
<td>The replacement value for '&lt;value&gt;' is invalid.</td>
</tr>
<tr>
<td>42815.S.171</td>
<td>The data type, length or value of arguments '&lt;value&gt;' and '&lt;value&gt;' is incompatible.</td>
</tr>
<tr>
<td>42818</td>
<td>Comparisons between '&lt;type&gt;' and '&lt;type&gt;' 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')</td>
</tr>
<tr>
<td>42820</td>
<td>The floating point literal '&lt;string&gt;' contains more than 30 characters.</td>
</tr>
<tr>
<td>42821</td>
<td>Columns of type '&lt;type&gt;' cannot hold values of type '&lt;type&gt;'.</td>
</tr>
<tr>
<td>42824</td>
<td>An operand of LIKE is not a string, or the first operand is not a column.</td>
</tr>
<tr>
<td>42831</td>
<td>'&lt;columnName&gt;' cannot be a column of a primary key or unique key because it can contain null values.</td>
</tr>
<tr>
<td>42831.S.1</td>
<td>'&lt;columnName&gt;' cannot be a column of a primary key because it can contain null values.</td>
</tr>
<tr>
<td>42834</td>
<td>SET NULL cannot be specified because FOREIGN KEY '&lt;key&gt;' cannot contain null values.</td>
</tr>
<tr>
<td>42837</td>
<td>ALTER TABLE '&lt;tableName&gt;' specified attributes for column '&lt;columnName&gt;' that are not compatible with the existing column.</td>
</tr>
<tr>
<td>42846</td>
<td>Cannot convert types '&lt;type&gt;' to '&lt;type&gt;'.</td>
</tr>
<tr>
<td>42877</td>
<td>A qualified column name '&lt;columnName&gt;' is not allowed in the ORDER BY clause.</td>
</tr>
<tr>
<td>42878</td>
<td>The ORDER BY clause of a SELECT UNION statement only supports unqualified column references and column position numbers. Other expressions are not currently supported.</td>
</tr>
<tr>
<td>42879</td>
<td>The ORDER BY clause may not contain column '&lt;columnName&gt;', since the query specifies DISTINCT and that column does not appear in the query result.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>4287A</td>
<td>The ORDER BY clause may not specify an expression, since the query specifies DISTINCT.</td>
</tr>
<tr>
<td>4287B</td>
<td>In this context, the ORDER BY clause may only specify a column number.</td>
</tr>
<tr>
<td>42884</td>
<td>No authorized routine named '&lt;routineName&gt;' of type '&lt;type&gt;' having compatible arguments was found.</td>
</tr>
<tr>
<td>42886</td>
<td>'&lt;value&gt;' parameter '&lt;value&gt;' requires a parameter marker '?'.</td>
</tr>
<tr>
<td>42894</td>
<td>DEFAULT value or IDENTITY attribute value is not valid for column '&lt;columnName&gt;'.</td>
</tr>
<tr>
<td>428C1</td>
<td>Only one identity column is allowed in a table.</td>
</tr>
<tr>
<td>428EK</td>
<td>The qualifier for a declared global temporary table name must be SESSION.</td>
</tr>
<tr>
<td>428C2</td>
<td>DELETE ROWS is not supported for ON '&lt;txnMode&gt;'.</td>
</tr>
<tr>
<td>428C3</td>
<td>Temporary table columns cannot be referenced by foreign keys.</td>
</tr>
<tr>
<td>428C4</td>
<td>Attempt to add temporary table, '&lt;txnMode&gt;', as a view dependency.</td>
</tr>
<tr>
<td>42903</td>
<td>Invalid use of an aggregate function.</td>
</tr>
<tr>
<td>42908</td>
<td>The CREATE VIEW statement does not include a column list.</td>
</tr>
<tr>
<td>42909</td>
<td>The CREATE TABLE statement does not include a column list.</td>
</tr>
<tr>
<td>42915</td>
<td>Foreign Key '&lt;key&gt;' is invalid because '&lt;value&gt;'.</td>
</tr>
<tr>
<td>42916</td>
<td>Synonym '&lt;synonym2&gt;' cannot be created for '&lt;synonym1&gt;' as it would result in a circular synonym chain.</td>
</tr>
<tr>
<td>42939</td>
<td>An object cannot be created with the schema name '&lt;schemaName&gt;'.</td>
</tr>
<tr>
<td>4293A</td>
<td>A role cannot be created with the name '&lt;authorizationID&gt;', the SYS prefix is reserved.</td>
</tr>
<tr>
<td>42962</td>
<td>Long column type column or parameter '&lt;columnName&gt;' not permitted in declared global temporary tables or procedure definitions.</td>
</tr>
<tr>
<td>42995</td>
<td>The requested function does not apply to global temporary tables.</td>
</tr>
<tr>
<td>42X01</td>
<td>Syntax error: &lt;error&gt;.</td>
</tr>
<tr>
<td>42X02</td>
<td>&lt;value&gt;.</td>
</tr>
<tr>
<td>42X03</td>
<td>Column name '&lt;columnName&gt;' is in more than one table in the FROM list.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>42X04</td>
<td>Column '&lt;columnName&gt;' 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 '&lt;columnName&gt;' is not a column in the target table.</td>
</tr>
<tr>
<td>42X05</td>
<td>Table/View '&lt;objectName&gt;' does not exist.</td>
</tr>
<tr>
<td>42X06</td>
<td>Too many result columns specified for table '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>42X07</td>
<td>Null is only allowed in a VALUES clause within an INSERT statement.</td>
</tr>
<tr>
<td>42X08</td>
<td>The constructor for class '&lt;className&gt;' cannot be used as an external virtual table because the class does not implement '&lt;constructorName&gt;'.</td>
</tr>
<tr>
<td>42X09</td>
<td>The table or alias name '&lt;tableName&gt;' is used more than once in the FROM list.</td>
</tr>
<tr>
<td>42X10</td>
<td>'&lt;tableName&gt;' is not an exposed table name in the scope in which it appears.</td>
</tr>
<tr>
<td>42X12</td>
<td>Column name '&lt;columnName&gt;' appears more than once in the CREATE TABLE statement.</td>
</tr>
<tr>
<td>42X13</td>
<td>Column name '&lt;columnName&gt;' appears more than once times in the column list of an INSERT statement.</td>
</tr>
<tr>
<td>42X14</td>
<td>'&lt;columnName&gt;' is not a column in table or VTI '&lt;value&gt;'.</td>
</tr>
<tr>
<td>42X15</td>
<td>Column name '&lt;columnName&gt;' appears in a statement without a FROM list.</td>
</tr>
<tr>
<td>42X16</td>
<td>Column name '&lt;columnName&gt;' appears multiple times in the SET clause of an UPDATE statement.</td>
</tr>
<tr>
<td>42X17</td>
<td>In the Properties list of a FROM clause, the value '&lt;value&gt;' is not valid as a joinOrder specification. Only the values FIXED and UNFIXED are valid.</td>
</tr>
<tr>
<td>42X19</td>
<td>The WHERE or HAVING clause or CHECK CONSTRAINT definition is a '&lt;value&gt;' expression. It must be a BOOLEAN expression.</td>
</tr>
<tr>
<td>42X20</td>
<td>Syntax error; integer literal expected.</td>
</tr>
<tr>
<td>42X23</td>
<td>Cursor &lt;cursorName&gt; is not updatable.</td>
</tr>
<tr>
<td>42X24</td>
<td>Column &lt;columnName&gt; is referenced in the HAVING clause but is not in the GROUP BY list.</td>
</tr>
<tr>
<td>42X25</td>
<td>The '&lt;functionName&gt;' function is not allowed on the type.</td>
</tr>
<tr>
<td>42X26</td>
<td>The class '&lt;className&gt;' for column '&lt;columnName&gt;' does not exist or is inaccessible. This can happen if the class is not public.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>42X28</td>
<td>Delete table '&lt;tableName&gt;' is not target of cursor '&lt;cursorName&gt;'.</td>
</tr>
<tr>
<td>42X29</td>
<td>Update table '&lt;tableName&gt;' is not the target of cursor '&lt;cursorName&gt;'.</td>
</tr>
<tr>
<td>42X30</td>
<td>Cursor '&lt;cursorName&gt;' not found. Verify that autocommit is OFF.</td>
</tr>
<tr>
<td>42X31</td>
<td>Column '&lt;columnName&gt;' is not in the FOR UPDATE list of cursor '&lt;cursorName&gt;'.</td>
</tr>
<tr>
<td>42X32</td>
<td>The number of columns in the derived column list must match the number of columns in table '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>42X33</td>
<td>The derived column list contains a duplicate column name '&lt;columnName&gt;'.</td>
</tr>
<tr>
<td>42X34</td>
<td>There is a ? parameter in the select list. This is not allowed.</td>
</tr>
<tr>
<td>42X35</td>
<td>It is not allowed for both operands of '&lt;value&gt;' to be ? parameters.</td>
</tr>
<tr>
<td>42X36</td>
<td>The '&lt;operator&gt;' operator is not allowed to take a ? parameter as an operand.</td>
</tr>
<tr>
<td>42X37</td>
<td>The unary '&lt;operator&gt;' operator is not allowed on the '&lt;type&gt;' type.</td>
</tr>
<tr>
<td>42X38</td>
<td>'SELECT *' only allowed in EXISTS and NOT EXISTS subqueries.</td>
</tr>
<tr>
<td>42X39</td>
<td>Subquery is only allowed to return a single column.</td>
</tr>
<tr>
<td>42X40</td>
<td>A NOT statement has an operand that is not boolean. The operand of NOT must evaluate to TRUE, FALSE, or UNKNOWN.</td>
</tr>
<tr>
<td>42X41</td>
<td>In the Properties clause of a FROM list, the property '&lt;propertyName&gt;' is not valid (the property was being set to '&lt;value&gt;').</td>
</tr>
<tr>
<td>42X42</td>
<td>Correlation name not allowed for column '&lt;columnName&gt;' because it is part of the FOR UPDATE list.</td>
</tr>
<tr>
<td>42X43</td>
<td>The ResultSetMetaData returned for the class/object '&lt;className&gt;' was null. In order to use this class as an external virtual table, the ResultSetMetaData cannot be null.</td>
</tr>
<tr>
<td>42X44</td>
<td>Invalid length '&lt;number&gt;' in column specification.</td>
</tr>
<tr>
<td>42X45</td>
<td>'&lt;type&gt;' is an invalid type for argument number &lt;value&gt; of &lt;value&gt;.</td>
</tr>
<tr>
<td>42X46</td>
<td>There are multiple functions named '&lt;functionName&gt;'. Use the full signature or the specific name.</td>
</tr>
<tr>
<td>42X47</td>
<td>There are multiple procedures named '&lt;procedureName&gt;'. Use the full signature or the specific name.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>42X48</td>
<td>Value '&lt;value&gt;' is not a valid precision for &lt;value&gt;.</td>
</tr>
<tr>
<td>42X49</td>
<td>Value '&lt;value&gt;' is not a valid integer literal.</td>
</tr>
<tr>
<td>42X50</td>
<td>No method was found that matched the method call <code>&lt;methodName&gt;.(&lt;value&gt;)</code>, 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.</td>
</tr>
<tr>
<td>42X51</td>
<td>The class '&lt;className&gt;' does not exist or is inaccessible. This can happen if the class is not public.</td>
</tr>
<tr>
<td>42X52</td>
<td>Calling method ('&lt;methodName&gt;') using a receiver of the Java primitive type '&lt;type&gt;' is not allowed.</td>
</tr>
<tr>
<td>42X53</td>
<td>The LIKE predicate can only have 'CHAR' or 'VARCHAR' operands. Type '&lt;type&gt;' is not permitted.</td>
</tr>
<tr>
<td>42X54</td>
<td>The Java method '&lt;methodName&gt;' has a ? as a receiver. This is not allowed.</td>
</tr>
<tr>
<td>42X55</td>
<td>Table name '&lt;tableName&gt;' should be the same as '&lt;value&gt;'.</td>
</tr>
<tr>
<td>42X56</td>
<td>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 '&lt;value&gt;'.</td>
</tr>
<tr>
<td>42X57</td>
<td>The getColumnCount() for external virtual table '&lt;tableName&gt;' returned an invalid value '&lt;value&gt;'. Valid values are greater than or equal to 1.</td>
</tr>
<tr>
<td>42X58</td>
<td>The number of columns on the left and right sides of the &lt;tableName&gt; must be the same.</td>
</tr>
<tr>
<td>42X59</td>
<td>The number of columns in each VALUES constructor must be the same.</td>
</tr>
<tr>
<td>42X60</td>
<td>Invalid value '&lt;value&gt;' for insertMode property specified for table '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>42X61</td>
<td>Types '&lt;type&gt;' and '&lt;type&gt;' are not &lt;value&gt; compatible.</td>
</tr>
<tr>
<td>42X62</td>
<td>'&lt;value&gt;' is not allowed in the '&lt;schemaName&gt;' schema.</td>
</tr>
<tr>
<td>42X63</td>
<td>The USING clause did not return any results. No parameters can be set.</td>
</tr>
<tr>
<td>42X64</td>
<td>In the Properties list, the invalid value '&lt;value&gt;' was specified for the useStatistics property. The only valid values are TRUE or FALSE.</td>
</tr>
<tr>
<td>42X65</td>
<td>Index '&lt;index&gt;' does not exist.</td>
</tr>
<tr>
<td>42X66</td>
<td>Column name '&lt;columnName&gt;' appears more than once in the CREATE INDEX statement.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>42X68</td>
<td>No field '&lt;fieldName&gt;' was found belonging to class '&lt;className&gt;'. It may be that the field exists, but it is not public, or that the class does not exist or is not public.</td>
</tr>
<tr>
<td>42X69</td>
<td>It is not allowed to reference a field ('&lt;fieldName&gt;') using a referencing expression of the Java primitive type '&lt;type&gt;'.</td>
</tr>
<tr>
<td>42X70</td>
<td>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 '&lt;value&gt;'.</td>
</tr>
<tr>
<td>42X71</td>
<td>Invalid data type '&lt;datatypeName&gt;' for column '&lt;columnName&gt;'.</td>
</tr>
<tr>
<td>42X72</td>
<td>No static field '&lt;fieldName&gt;' was found belonging to class '&lt;className&gt;'. The field might exist, but it is not public and/or static, or the class does not exist or the class is not public.</td>
</tr>
<tr>
<td>42X73</td>
<td>Method resolution for signature &lt;value&gt;.&lt;value&gt;(&lt;value&gt;) was ambiguous. (No single maximally specific method.)</td>
</tr>
<tr>
<td>42X74</td>
<td>Invalid CALL statement syntax.</td>
</tr>
<tr>
<td>42X75</td>
<td>No constructor was found with the signature &lt;value&gt;(&lt;value&gt;). It may be that the parameter types are not method invocation convertible.</td>
</tr>
<tr>
<td>42X76</td>
<td>At least one column, '&lt;columnName&gt;', in the primary key being added is nullable. All columns in a primary key must be non-nullable.</td>
</tr>
<tr>
<td>42X77</td>
<td>Column position '&lt;columnPosition&gt;' is out of range for the query expression.</td>
</tr>
<tr>
<td>42X78</td>
<td>Column '&lt;columnName&gt;' is not in the result of the query expression.</td>
</tr>
<tr>
<td>42X79</td>
<td>Column name '&lt;columnName&gt;' appears more than once in the result of the query expression.</td>
</tr>
<tr>
<td>42X80</td>
<td>VALUES clause must contain at least one element. Empty elements are not allowed.</td>
</tr>
<tr>
<td>42X81</td>
<td>A query expression must return at least one column.</td>
</tr>
<tr>
<td>42X82</td>
<td>The USING clause returned more than one row. Only single-row ResultSets are permissible.</td>
</tr>
<tr>
<td>42X83</td>
<td>The constraints on column '&lt;columnName&gt;' require that it be both nullable and not nullable.</td>
</tr>
<tr>
<td>42X84</td>
<td>Index '&lt;index&gt;' was created to enforce constraint '&lt;constraintName&gt;'. It can only be dropped by dropping the constraint.</td>
</tr>
<tr>
<td>42X85</td>
<td>Constraint '&lt;constraintName&gt;' is required to be in the same schema as table '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>42X86</td>
<td>ALTER TABLE failed. There is no constraint '&lt;constraintName&gt;' on table '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
</tbody>
</table>
| 42X87    | At least one result expression (THEN or ELSE) of the '<expression>' expression must not be a '?'.
<p>| 42X88    | A conditional has a non-Boolean operand. The operand of a conditional must evaluate to TRUE, FALSE, or UNKNOWN. |
| 42X89    | Types '&lt;type&gt;' and '&lt;type&gt;' are not type compatible. Neither type is assignable to the other type. |
| 42X90    | More than one primary key constraint specified for table '&lt;tableName&gt;'. |
| 42X91    | Constraint name '&lt;constraintName&gt;' appears more than once in the CREATE TABLE statement. |
| 42X92    | Column name '&lt;columnName&gt;' appears more than once in a constraint's column list. |
| 42X93    | Table '&lt;tableName&gt;' contains a constraint definition with column '&lt;columnName&gt;' which is not in the table. |
| 42X94    | '&lt;value&gt;' '&lt;value&gt;' does not exist. |
| 42X96    | The database class path contains an unknown jar file '&lt;fileName&gt;'. |
| 42X98    | Parameters are not allowed in a VIEW definition. |
| 42X99    | Parameters are not allowed in a TABLE definition. |
| 42XA0    | The generation clause for column '&lt;columnName&gt;' has data type '&lt;datatypeName&gt;', which cannot be assigned to the column's declared data type. |
| 42XA1    | The generation clause for column '&lt;columnName&gt;' contains an aggregate. This is not allowed. |
| 42XA2    | '&lt;value&gt;' cannot appear in a GENERATION CLAUSE because it may return unreliable results. |
| 42XA3    | You may not override the value of generated column '&lt;columnName&gt;'. |
| 42XA4    | The generation clause for column '&lt;columnName&gt;' references other generated columns. This is not allowed. |
| 42XA5    | Routine '&lt;routineName&gt;' may issue SQL and therefore cannot appear in a GENERATION CLAUSE. |
| 42XA6    | '&lt;columnName&gt;' 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. |</p>
<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>42XA7</td>
<td>'&lt;columnName&gt;' is a generated column. You cannot change its default value.</td>
</tr>
<tr>
<td>42XA8</td>
<td>You cannot rename '&lt;columnName&gt;' because it is referenced by the generation clause of column '&lt;columnName&gt;'.</td>
</tr>
<tr>
<td>42XA9</td>
<td>Column '&lt;columnName&gt;' needs an explicit datatype. The datatype can be omitted only for columns with generation clauses.</td>
</tr>
<tr>
<td>42XAA</td>
<td>The NEW value of generated column '&lt;columnName&gt;' is mentioned in the BEFORE action of a trigger. This is not allowed.</td>
</tr>
<tr>
<td>42XAB</td>
<td>NOT NULL is allowed only if you explicitly declare a datatype.</td>
</tr>
<tr>
<td>42XAC</td>
<td>'INCREMENT BY' value can not be zero.</td>
</tr>
<tr>
<td>42XAE</td>
<td>'&lt;argName&gt;' value out of range of datatype '&lt;datatypeName&gt;'. Must be between '&lt;minValue&gt;' and '&lt;maxValue&gt;'.</td>
</tr>
<tr>
<td>42XAF</td>
<td>Invalid 'MINVALUE' value '&lt;minValue&gt;'. Must be smaller than 'MAXVALUE: &lt;maxValue&gt;'.</td>
</tr>
<tr>
<td>42XAG</td>
<td>Invalid 'START WITH' value '&lt;startValue&gt;'. Must be between '&lt;minValue&gt;' and '&lt;maxValue&gt;'.</td>
</tr>
<tr>
<td>42XAH</td>
<td>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.</td>
</tr>
<tr>
<td>42XAI</td>
<td>The statement references the following sequence more than once: '&lt;sequenceName&gt;'.</td>
</tr>
<tr>
<td>42XAJ</td>
<td>The CREATE SEQUENCE statement has a redundant '&lt;clauseName&gt;' clause.</td>
</tr>
<tr>
<td>42Y00</td>
<td>Class '&lt;className&gt;' does not implement com.splicemachine.db.iapi.db.AggregateDefinition and thus cannot be used as an aggregate expression.</td>
</tr>
<tr>
<td>42Y01</td>
<td>Constraint '&lt;constraintName&gt;' is invalid.</td>
</tr>
<tr>
<td>42Y03.S.0</td>
<td>'&lt;statement&gt;' is not recognized as a function or procedure.</td>
</tr>
<tr>
<td>42Y03.S.1</td>
<td>'&lt;statement&gt;' is not recognized as a procedure.</td>
</tr>
<tr>
<td>42Y03.S.2</td>
<td>'&lt;statement&gt;' is not recognized as a function.</td>
</tr>
<tr>
<td>42Y04</td>
<td>Cannot create a procedure or function with EXTERNAL NAME '&lt;name&gt;' because it is not a list separated by periods. The expected format is &lt;full java path&gt;.&lt;method name&gt;.</td>
</tr>
<tr>
<td>42Y05</td>
<td>There is no Foreign Key named '&lt;key&gt;'.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>42Y07</td>
<td>Schema '&lt;schemaName&gt;' does not exist</td>
</tr>
<tr>
<td>42Y08</td>
<td>Foreign key constraints are not allowed on system tables.</td>
</tr>
<tr>
<td>42Y09</td>
<td>Void methods are only allowed within a CALL statement.</td>
</tr>
<tr>
<td>42Y10</td>
<td>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.</td>
</tr>
<tr>
<td>42Y11</td>
<td>A join specification is required with the '&lt;clauseName&gt;' clause.</td>
</tr>
<tr>
<td>42Y12</td>
<td>The ON clause of a JOIN is a '&lt;expressionType&gt;' expression. It must be a BOOLEAN expression.</td>
</tr>
<tr>
<td>42Y13</td>
<td>Column name '&lt;columnName&gt;' appears more than once in the CREATE VIEW statement.</td>
</tr>
<tr>
<td>42Y16</td>
<td>No public static method '&lt;methodName&gt;' was found in class '&lt;className&gt;'. The method might exist, but it is not public, or it is not static.</td>
</tr>
<tr>
<td>42Y22</td>
<td>Aggregate &lt;aggregateType&gt; cannot operate on type &lt;type&gt;.</td>
</tr>
<tr>
<td>42Y23</td>
<td>Incorrect JDBC type info returned for column &lt;columnName&gt;.</td>
</tr>
<tr>
<td>42Y24</td>
<td>View '&lt;viewName&gt;' is not updatable. (Views are currently not updatable.)</td>
</tr>
<tr>
<td>42Y25</td>
<td>'&lt;tableName&gt;' is a system table. Users are not allowed to modify the contents of this table.</td>
</tr>
<tr>
<td>42Y26</td>
<td>Aggregates are not allowed in the GROUP BY list.</td>
</tr>
<tr>
<td>42Y27</td>
<td>Parameters are not allowed in the trigger action.</td>
</tr>
<tr>
<td>42Y29</td>
<td>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.</td>
</tr>
<tr>
<td>42Y30</td>
<td>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.</td>
</tr>
<tr>
<td>42Y32</td>
<td>Aggregator class '&lt;className&gt;' for aggregate '&lt;aggregateName&gt;' on type &lt;type&gt; does not implement com.splicemachine.db.iapi.sql.execute.ExecAggregator.</td>
</tr>
<tr>
<td>42Y33</td>
<td>Aggregate &lt;aggregateName&gt; contains one or more aggregates.</td>
</tr>
<tr>
<td>42Y34</td>
<td>Column name '&lt;columnName&gt;' matches more than one result column in table '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>42Y35</td>
<td>Column reference '&lt;reference&gt;' is invalid. When the SELECT list contains at least one aggregate then all entries must be valid aggregate expressions.</td>
</tr>
<tr>
<td>42Y36</td>
<td>Column reference '&lt;reference&gt;' 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.</td>
</tr>
<tr>
<td>42Y37</td>
<td>'&lt;value&gt;' is a Java primitive and cannot be used with this operator.</td>
</tr>
<tr>
<td>42Y38</td>
<td>insertMode = replace is not permitted on an insert where the target table, '&lt;tableName&gt;', is referenced in the SELECT.</td>
</tr>
<tr>
<td>42Y39</td>
<td>'&lt;value&gt;' may not appear in a CHECK CONSTRAINT definition because it may return non-deterministic results.</td>
</tr>
<tr>
<td>42Y40</td>
<td>'&lt;value&gt;' appears multiple times in the UPDATE OF column list for trigger '&lt;triggerName&gt;'.</td>
</tr>
<tr>
<td>42Y41</td>
<td>'&lt;value&gt;' cannot be directly invoked via EXECUTE STATEMENT because it is part of a trigger.</td>
</tr>
<tr>
<td>42Y42</td>
<td>Scale '&lt;scaleValue&gt;' is not a valid scale for a '&lt;value&gt;'.</td>
</tr>
<tr>
<td>42Y43</td>
<td>Scale '&lt;scaleValue&gt;' is not a valid scale with precision of '&lt;precision&gt;'.</td>
</tr>
<tr>
<td>42Y44</td>
<td>Invalid key '&lt;key&gt;' specified in the Properties list of a FROM list. The case-sensitive keys that are currently supported are '&lt;key&gt;'.</td>
</tr>
<tr>
<td>42Y45</td>
<td>VTI '&lt;value&gt;' cannot be bound because it is a special trigger VTI and this statement is not part of a trigger action or WHEN clause.</td>
</tr>
<tr>
<td>42Y46</td>
<td>Invalid Properties list in FROM list. There is no index '&lt;index&gt;' on table '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>42Y47</td>
<td>Invalid Properties list in FROM list. The hint useSpark needs (true/false) and does not support '&lt;value&gt;'.</td>
</tr>
<tr>
<td>42Y48</td>
<td>Invalid Properties list in FROM list. Either there is no named constraint '&lt;constraintName&gt;' on table '&lt;tableName&gt;' or the constraint does not have a backing index.</td>
</tr>
<tr>
<td>42Y49</td>
<td>Multiple values specified for property key '&lt;key&gt;'.</td>
</tr>
<tr>
<td>42Y50</td>
<td>Properties list for table '&lt;tableName&gt;' may contain values for index or for constraint but not both.</td>
</tr>
<tr>
<td>42Y55</td>
<td>'&lt;value&gt;' cannot be performed on '&lt;value&gt;' because it does not exist.</td>
</tr>
<tr>
<td>42Y56</td>
<td>Invalid join strategy '&lt;strategyValue&gt;' specified in Properties list on table '&lt;tableName&gt;'. The currently supported values for a join strategy are: &lt;supportedStrategyNames&gt;.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>42Y58</td>
<td>NumberFormatException occurred when converting value '&lt;value&gt;' for optimizer override '&lt;value&gt;'.</td>
</tr>
<tr>
<td>42Y59</td>
<td>Invalid value, '&lt;value&gt;', specified for hashInitialCapacity override. Value must be greater than 0.</td>
</tr>
<tr>
<td>42Y60</td>
<td>Invalid value, '&lt;value&gt;', specified for hashLoadFactor override. Value must be greater than 0.0 and less than or equal to 1.0.</td>
</tr>
<tr>
<td>42Y61</td>
<td>Invalid value, '&lt;value&gt;', specified for hashMaxCapacity override. Value must be greater than 0.</td>
</tr>
<tr>
<td>42Y62</td>
<td>'&lt;statement&gt;' is not allowed on '&lt;viewName&gt;' because it is a view.</td>
</tr>
<tr>
<td>42Y63</td>
<td>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 '&lt;index&gt;'. Use the 'index' optimizer override to specify such an index or the heap on table '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>42Y64</td>
<td>bulkFetch value of '&lt;value&gt;' is invalid. The minimum value for bulkFetch is 1.</td>
</tr>
<tr>
<td>42Y65</td>
<td>bulkFetch is not permitted on '&lt;joinType&gt;' joins.</td>
</tr>
<tr>
<td>42Y66</td>
<td>bulkFetch is not permitted on updatable cursors.</td>
</tr>
<tr>
<td>42Y67</td>
<td>Schema '&lt;schemaName&gt;' cannot be dropped.</td>
</tr>
<tr>
<td>42Y69</td>
<td>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.</td>
</tr>
<tr>
<td>42Y70</td>
<td>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.</td>
</tr>
<tr>
<td>42Y71</td>
<td>System function or procedure '&lt;procedureName&gt;' cannot be dropped.</td>
</tr>
<tr>
<td>42Y82</td>
<td>System generated stored prepared statement '&lt;statement&gt;' that cannot be dropped using DROP STATEMENT. It is part of a trigger.</td>
</tr>
<tr>
<td>42Y83</td>
<td>An untyped null is not permitted as an argument to aggregate '&lt;aggregateName&gt;'. Please cast the null to a suitable type.</td>
</tr>
<tr>
<td>42Y84</td>
<td>'&lt;value&gt;' may not appear in a DEFAULT definition.</td>
</tr>
<tr>
<td>42Y85</td>
<td>The DEFAULT keyword is only allowed in a VALUES clause when the VALUES clause appears within an INSERT statement.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>42Y90</td>
<td>FOR UPDATE is not permitted in this type of statement.</td>
</tr>
<tr>
<td>42Y91</td>
<td>The USING clause is not permitted in an EXECUTE STATEMENT for a trigger action.</td>
</tr>
<tr>
<td>42Y92</td>
<td>&lt;triggerName&gt; triggers may only reference &lt;value&gt; transition variables/tables.</td>
</tr>
<tr>
<td>42Y93</td>
<td>Illegal REFERENCING clause: only one name is permitted for each type of transition variable/table.</td>
</tr>
<tr>
<td>42Y94</td>
<td>An AND or OR has a non-boolean operand. The operands of AND and OR must evaluate to TRUE, FALSE, or UNKNOWN.</td>
</tr>
<tr>
<td>42Y95</td>
<td>The '&lt;operatorName&gt;' operator with a left operand type of '&lt;operandType&gt;' and a right operand type of '&lt;operandType&gt;' is not supported.</td>
</tr>
<tr>
<td>42Y96</td>
<td>Invalid Sort Strategy: '&lt;sortStrategy&gt;'.</td>
</tr>
<tr>
<td>42Y97</td>
<td>Invalid escape character at line '&lt;lineNumber&gt;', column '&lt;columnName&gt;'.</td>
</tr>
<tr>
<td>42Z02</td>
<td>Multiple DISTINCT aggregates are not supported at this time.</td>
</tr>
<tr>
<td>42Z07</td>
<td>Aggregates are not permitted in the ON clause.</td>
</tr>
<tr>
<td>42Z08</td>
<td>Bulk insert replace is not permitted on '&lt;value&gt;' because it has an enabled trigger (&lt;value&gt;).</td>
</tr>
<tr>
<td>42Z15</td>
<td>Invalid type specified for column '&lt;columnName&gt;'. The type of a column may not be changed.</td>
</tr>
<tr>
<td>42Z16</td>
<td>Only columns of type VARCHAR, CLOB, and BLOB may have their length altered.</td>
</tr>
<tr>
<td>42Z17</td>
<td>Invalid length specified for column '&lt;columnName&gt;'. Length must be greater than the current column length.</td>
</tr>
<tr>
<td>42Z18</td>
<td>Column '&lt;columnName&gt;' is part of a foreign key constraint '&lt;constraintName&gt;'. To alter the length of this column, you should drop the constraint first, perform the ALTER TABLE, and then recreate the constraint.</td>
</tr>
<tr>
<td>42Z19</td>
<td>Column '&lt;columnName&gt;' is being referenced by at least one foreign key constraint '&lt;constraintName&gt;'. To alter the length of this column, you should drop referencing constraints, perform the ALTER TABLE and then recreate the constraints.</td>
</tr>
<tr>
<td>42Z20</td>
<td>Column '&lt;columnName&gt;' cannot be made nullable. It is part of a primary key or unique constraint, which cannot have any nullable columns.</td>
</tr>
<tr>
<td>42Z20.S.1</td>
<td>Column '&lt;columnName&gt;' cannot be made nullable. It is part of a primary key, which cannot have any nullable columns.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>42Z21</td>
<td>Invalid increment specified for identity focr column '&lt;columnName&gt;'. Increment cannot be zero.</td>
</tr>
<tr>
<td>42Z22</td>
<td>Invalid type specified for identity column '&lt;columnName&gt;'. The only valid types for identity columns are BIGINT, INT and SMALLINT.</td>
</tr>
<tr>
<td>42Z23</td>
<td>Attempt to modify an identity column '&lt;columnName&gt;'.</td>
</tr>
<tr>
<td>42Z24</td>
<td>Overflow occurred in identity value for column '&lt;columnName&gt;' in table '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>42Z25</td>
<td>INTERNAL ERROR identity counter. Update was called without arguments with current value = NULL.</td>
</tr>
<tr>
<td>42Z26</td>
<td>A column, '&lt;columnName&gt;', with an identity default cannot be made nullable.</td>
</tr>
<tr>
<td>42Z27</td>
<td>A nullable column, '&lt;columnName&gt;', cannot be modified to have identity default.</td>
</tr>
<tr>
<td>42Z50</td>
<td>INTERNAL ERROR: Unable to generate code for &lt;value&gt;.</td>
</tr>
<tr>
<td>42Z53</td>
<td>INTERNAL ERROR: Type of activation to generate for node choice &lt;value&gt; is unknown.</td>
</tr>
<tr>
<td>42Z60</td>
<td>&lt;value&gt; not allowed unless database property &lt;propertyName&gt; has value '&lt;value&gt;'.</td>
</tr>
<tr>
<td>42Z70</td>
<td>Binding directly to an XML value is not allowed; try using XMLPARSE.</td>
</tr>
<tr>
<td>42Z71</td>
<td>XML values are not allowed in top-level result sets; try using XMLSERIALIZE.</td>
</tr>
<tr>
<td>42Z72</td>
<td>Missing SQL/XML keyword(s) '&lt;keywords&gt;' at line &lt;lineNumber&gt;, column &lt;columnNumber&gt;.</td>
</tr>
<tr>
<td>42Z73</td>
<td>Invalid target type for XMLSERIALIZE: '&lt;typeName&gt;'.</td>
</tr>
<tr>
<td>42Z74</td>
<td>XML feature not supported: '&lt;featureName&gt;'.</td>
</tr>
<tr>
<td>42Z75</td>
<td>XML query expression must be a string literal.</td>
</tr>
<tr>
<td>42Z76</td>
<td>Multiple XML context items are not allowed.</td>
</tr>
<tr>
<td>42Z77</td>
<td>Context item must have type 'XML'; '&lt;value&gt;' is not allowed.</td>
</tr>
<tr>
<td>42Z79</td>
<td>Unable to determine the parameter type for XMLPARSE; try using a CAST.</td>
</tr>
<tr>
<td>42Z90</td>
<td>Class '&lt;className&gt;' does not return an updatable ResultSet.</td>
</tr>
<tr>
<td>42Z91</td>
<td>subquery</td>
</tr>
<tr>
<td>42Z92</td>
<td>repeatable read</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>42Z93</td>
<td>Constraints '&lt;constraintName&gt;' and '&lt;constraintName&gt;' have the same set of columns, which is not allowed.</td>
</tr>
<tr>
<td>42Z97</td>
<td>Renaming column '&lt;columnName&gt;' will cause check constraint '&lt;constraintName&gt;' to break.</td>
</tr>
<tr>
<td>42Z99</td>
<td>String or Hex literal cannot exceed 64K.</td>
</tr>
<tr>
<td>42Z9A</td>
<td>read uncommitted</td>
</tr>
<tr>
<td>42Z9B</td>
<td>The external virtual table interface does not support BLOB or CLOB columns. '&lt;value&gt;' column '&lt;value&gt;'.</td>
</tr>
<tr>
<td>42Z9D.S.1</td>
<td>Procedures that modify SQL data are not allowed in BEFORE triggers.</td>
</tr>
<tr>
<td>42Z9D</td>
<td>'&lt;statement&gt;' statements are not allowed in '&lt;triggerName&gt;' triggers.</td>
</tr>
<tr>
<td>42Z9E</td>
<td>Constraint '&lt;constraintName&gt;' is not a &lt;value&gt; constraint.</td>
</tr>
<tr>
<td>42Z9F</td>
<td>Too many indexes (&lt;index&gt;) on the table &lt;tableName&gt;. The limit is &lt;number&gt;.</td>
</tr>
<tr>
<td>42ZA0</td>
<td>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.</td>
</tr>
<tr>
<td>42ZA1</td>
<td>Invalid SQL in Batch: '&lt;batch&gt;'.</td>
</tr>
<tr>
<td>42ZA2</td>
<td>Operand of LIKE predicate with type &lt;type&gt; and collation &lt;value&gt; is not compatible with LIKE pattern operand with type &lt;type&gt; and collation &lt;value&gt;.</td>
</tr>
<tr>
<td>42ZA3</td>
<td>The table will have collation type &lt;type&gt; which is different than the collation of the schema &lt;type&gt; hence this operation is not supported.</td>
</tr>
<tr>
<td>42ZB1</td>
<td>Parameter style SPLICE_JDBC_RESULT_SET is only allowed for table functions.</td>
</tr>
<tr>
<td>42ZB2</td>
<td>Table functions can only have parameter style SPLICE_JDBC_RESULT_SET.</td>
</tr>
<tr>
<td>42ZB3</td>
<td>XML is not allowed as the datatype of a user-defined aggregate or of a column returned by a table function.</td>
</tr>
<tr>
<td>42ZB4</td>
<td>'&lt;schemaName&gt;.'&lt;functionName&gt;' does not identify a table function.</td>
</tr>
<tr>
<td>42ZB5</td>
<td>Class '&lt;className&gt;' implements VTICosting but does not provide a public, no-arg constructor.</td>
</tr>
<tr>
<td>42ZB6</td>
<td>A scalar value is expected, not a row set returned by a table function.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>42ZC0</td>
<td>Window '&lt;windowName&gt;' is not defined.</td>
</tr>
<tr>
<td>42ZC1</td>
<td>Only one window is supported.</td>
</tr>
<tr>
<td>42ZC2</td>
<td>Window function is illegal in this context: '&lt;clauseName&gt;' clause</td>
</tr>
<tr>
<td>42ZC3</td>
<td>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: '&lt;aggregateName&gt;'</td>
</tr>
<tr>
<td>42ZC4</td>
<td>User defined aggregate '&lt;schemaName&gt;.''&lt;aggregateName&gt;' is bound to external class '&lt;className&gt;'. The parameter types of that class could not be resolved.</td>
</tr>
<tr>
<td>42ZC6</td>
<td>User defined aggregate '&lt;schemaName&gt;.''&lt;aggregateName&gt;' was declared to have this input Java type: '&lt;javaDataType&gt;'. This does not extend the following actual bounding input Java type: '&lt;javaDataType&gt;'.</td>
</tr>
<tr>
<td>42ZC7</td>
<td>User defined aggregate '&lt;schemaName&gt;.''&lt;aggregateName&gt;' was declared to have this return Java type: '&lt;javaDataType&gt;'. This does not extend the following actual bounding return Java type: '&lt;javaDataType&gt;'.</td>
</tr>
<tr>
<td>42ZC8</td>
<td>Implementing class '&lt;className&gt;' for user defined aggregate '&lt;schemaName&gt;.''&lt;aggregateName&gt;' could not be instantiated or was malformed. Detailed message follows: '&lt;detailedMessage&gt;'</td>
</tr>
<tr>
<td>43001</td>
<td>The truncate function was provided a null operand.</td>
</tr>
<tr>
<td>43002</td>
<td>The truncate function was provided an operand which it does not know how to handle: '&lt;operand&gt;'. It requires a DATE, TIMESTAMP, INTEGER or DECIMAL type.</td>
</tr>
<tr>
<td>43003</td>
<td>The truncate function expects a right-side argument of type CHAR for an operand of type DATE or TIMESTAMP but got: '&lt;truncValue&gt;'.</td>
</tr>
<tr>
<td>43004</td>
<td>The truncate function expects a right-side argument of type INTEGER for an operand of type DECIMAL but got: '&lt;truncValue&gt;'.</td>
</tr>
<tr>
<td>43005</td>
<td>The truncate function got an invalid right-side trunc value for operand type DATE: '&lt;truncValue&gt;'.</td>
</tr>
<tr>
<td>43006</td>
<td>The truncate function got an unknown right-side trunc value for operand type '&lt;operand&gt;': '&lt;truncValue&gt;'. Acceptable values are: '&lt;acceptableValues&gt;'.</td>
</tr>
<tr>
<td>44001</td>
<td>'&lt;dateOrTimestamp&gt;s cannot be multiplied or divided. The operation is undefined.</td>
</tr>
<tr>
<td>44002</td>
<td>'&lt;dateOrTimestamp&gt;s cannot be added. The operation is undefined.</td>
</tr>
<tr>
<td>44003</td>
<td>Timestamp '&lt;dateOrTimestamp&gt;' is out of range (~ from 0000-00-00 00:00:00 GMT to 9999-12-31 23:59:59 GMT).</td>
</tr>
</tbody>
</table>
Error Class 42: Syntax Error or Access Rule Violation
Error Class 57: DRDA Network Protocol: Execution Failure

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>57017.C</td>
<td>There is no available conversion for the source code page, <code>&lt;codePage&gt;</code>, to the target code page, <code>&lt;codePage&gt;</code>. The connection has been terminated.</td>
</tr>
</tbody>
</table>
Error Class 58: DRDA Network Protocol: Protocol Error

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>58009.C.10</td>
<td>Network protocol exception: only one of the VCM, VCS length can be greater than 0. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.11</td>
<td>The connection was terminated because the encoding is not supported.</td>
</tr>
<tr>
<td>58009.C.12</td>
<td>Network protocol exception: actual code point, <code>&lt;value&gt;</code>, does not match expected code point, <code>&lt;value&gt;</code>. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.13</td>
<td>Network protocol exception: DDM collection contains less than 4 bytes of data. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.14</td>
<td>Network protocol exception: collection stack not empty at end of same id chain parse. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.15</td>
<td>Network protocol exception: DSS length not 0 at end of same id chain parse. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.16</td>
<td>Network protocol exception: DSS chained with same id at end of same id chain parse. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.17</td>
<td>Network protocol exception: end of stream prematurely reached while reading InputStream, parameter #&lt;value&gt;. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.18</td>
<td>Network protocol exception: invalid FDOCA LID. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.19</td>
<td>Network protocol exception: SECTKN was not returned. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.20</td>
<td>Network protocol exception: only one of NVCM, NVCS can be non-null. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.21</td>
<td>Network protocol exception: SCLDTA length, <code>&lt;length&gt;</code>, is invalid for RDBNAM. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.7</td>
<td>Network protocol exception: SCLDTA length, <code>&lt;length&gt;</code>, is invalid for RDBCOLID. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.8</td>
<td>Network protocol exception: SCLDTA length, <code>&lt;length&gt;</code>, is invalid for PKGID. The connection has been terminated.</td>
</tr>
<tr>
<td>58009.C.9</td>
<td>Network protocol exception: PKGNAMCSN length, <code>&lt;length&gt;</code>, is invalid at SQLAM <code>&lt;value&gt;</code>. The connection has been terminated.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>58010.C</td>
<td>A network protocol error was encountered. A connection could not be established because the manager <code>&lt;value&gt;</code> at level <code>&lt;value&gt;</code> is not supported by the server.</td>
</tr>
<tr>
<td>58014.C</td>
<td>The DDM command 0x<code>&lt;value&gt;</code> is not supported. The connection has been terminated.</td>
</tr>
<tr>
<td>58015.C</td>
<td>The DDM object 0x<code>&lt;value&gt;</code> is not supported. The connection has been terminated.</td>
</tr>
<tr>
<td>58016.C</td>
<td>The DDM parameter 0x<code>&lt;value&gt;</code> is not supported. The connection has been terminated.</td>
</tr>
<tr>
<td>58017.C</td>
<td>The DDM parameter value 0x<code>&lt;value&gt;</code> is not supported. An input host variable may not be within the range the server supports. The connection has been terminated.</td>
</tr>
</tbody>
</table>
Error Class XBCA: CacheService

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XBCA0.S</td>
<td>Cannot create new object with key &lt;key&gt; in &lt;cache&gt; cache. The object already exists in the cache.</td>
</tr>
</tbody>
</table>
## Error Class XBCM: ClassManager

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XBCM1.S</td>
<td>Java linkage error thrown during load of generated class <code>&lt;className&gt;</code>.</td>
</tr>
<tr>
<td>XBCM2.S</td>
<td>Cannot create an instance of generated class <code>&lt;className&gt;</code>.</td>
</tr>
<tr>
<td>XBCM3.S</td>
<td>Method <code>&lt;methodName&gt;</code>() does not exist in generated class <code>&lt;className&gt;</code>.</td>
</tr>
<tr>
<td>XBCM4.S</td>
<td>Java class file format limit(s) exceeded: <code>&lt;value&gt;</code> in generated class <code>&lt;className&gt;</code>.</td>
</tr>
<tr>
<td>XBCM5.S</td>
<td>This operation requires that the JVM level be at least <code>&lt;vmLevel&gt;</code>.</td>
</tr>
</tbody>
</table>
## Error Class XBCX: Cryptography

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XBCX0.S</td>
<td>Exception from Cryptography provider. See next exception for details.</td>
</tr>
<tr>
<td>XBCX1.S</td>
<td>Initializing cipher with illegal mode, must be either ENCRYPT or DECRYPT.</td>
</tr>
<tr>
<td>XBCX2.S</td>
<td>Initializing cipher with a boot password that is too short. The password must be at least <code>&lt;number&gt;</code> characters long.</td>
</tr>
<tr>
<td>XBCX5.S</td>
<td>Cannot change boot password to null.</td>
</tr>
<tr>
<td>XBCX6.S</td>
<td>Cannot change boot password to a non-string serializable type.</td>
</tr>
<tr>
<td>XBCX7.S</td>
<td>Wrong format for changing boot password. Format must be : old_boot_password, new_boot_password.</td>
</tr>
<tr>
<td>XBCX8.S</td>
<td>Cannot change boot password for a non-encrypted database.</td>
</tr>
<tr>
<td>XBCX9.S</td>
<td>Cannot change boot password for a read-only database.</td>
</tr>
<tr>
<td>XBCXA.S</td>
<td>Wrong boot password.</td>
</tr>
<tr>
<td>XBCXB.S</td>
<td>Bad encryption padding '&lt;value&gt;' or padding not specified. 'NoPadding' must be used.</td>
</tr>
<tr>
<td>XBCXC.S</td>
<td>Encryption algorithm '&lt;algorithmName&gt;' does not exist. Please check that the chosen provider '&lt;providerName&gt;' supports this algorithm.</td>
</tr>
<tr>
<td>XBCXD.S</td>
<td>The encryption algorithm cannot be changed after the database is created.</td>
</tr>
<tr>
<td>XBCXE.S</td>
<td>The encryption provider cannot be changed after the database is created.</td>
</tr>
<tr>
<td>XBCXF.S</td>
<td>The class '&lt;className&gt;' representing the encryption provider cannot be found.</td>
</tr>
<tr>
<td>XBCXG.S</td>
<td>The encryption provider '&lt;providerName&gt;' does not exist.</td>
</tr>
<tr>
<td>XBCXH.S</td>
<td>The encryptionAlgorithm '&lt;algorithmName&gt;' is not in the correct format. The correct format is algorithm/feedbackMode/NoPadding.</td>
</tr>
<tr>
<td>XBCXI.S</td>
<td>The feedback mode '&lt;mode&gt;' is not supported. Supported feedback modes are CBC, CFB, OFB and ECB.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XBCXJ.S</td>
<td>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.</td>
</tr>
<tr>
<td>XBCXK.S</td>
<td>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.</td>
</tr>
<tr>
<td>XBCXL.S</td>
<td>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.</td>
</tr>
<tr>
<td>XBCXM.S</td>
<td>The length of the external encryption key must be an even number.</td>
</tr>
<tr>
<td>XBCXN.S</td>
<td>The external encryption key contains one or more illegal characters. Allowed characters for a hexadecimal number are 0-9, a-f and A-F.</td>
</tr>
<tr>
<td>XBCXO.S</td>
<td>Cannot encrypt the database when there is a global transaction in the prepared state.</td>
</tr>
<tr>
<td>XBCXP.S</td>
<td>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.</td>
</tr>
<tr>
<td>XBCXQ.S</td>
<td>Cannot configure a read-only database for encryption.</td>
</tr>
<tr>
<td>XBCXR.S</td>
<td>Cannot re-encrypt a read-only database with a new boot password or an external encryption key.</td>
</tr>
<tr>
<td>XBCXS.S</td>
<td>Cannot configure a database for encryption, when database is in the log archive mode.</td>
</tr>
<tr>
<td>XBCXT.S</td>
<td>Cannot re-encrypt a database with a new boot password or an external encryption key, when database is in the log archive mode.</td>
</tr>
<tr>
<td>XBCXU.S</td>
<td>Encryption of an un-encrypted database failed: &lt;failureMessage&gt;</td>
</tr>
<tr>
<td>XBCXV.S</td>
<td>Encryption of an encrypted database with a new key or a new password failed: &lt;failureMessage&gt;</td>
</tr>
<tr>
<td>XBCXW.S</td>
<td>The message digest algorithm '&lt;algorithmName&gt;' 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.</td>
</tr>
</tbody>
</table>
# Error Class XBM: Monitor

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XBM01.D</td>
<td>Startup failed due to an exception. See next exception for details.</td>
</tr>
<tr>
<td>XBM02.D</td>
<td>Startup failed due to missing functionality for <code>&lt;value&gt;</code>. Please ensure your classpath includes the correct Splice software.</td>
</tr>
<tr>
<td>XBM05.D</td>
<td>Startup failed due to missing product version information for <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XBM06.D</td>
<td>Startup failed. An encrypted database cannot be accessed without the correct boot password.</td>
</tr>
<tr>
<td>XBM07.D</td>
<td>Startup failed. Boot password must be at least 8 bytes long.</td>
</tr>
<tr>
<td>XBM08.D</td>
<td>Could not instantiate <code>&lt;value&gt;</code> StorageFactory class <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XBM0A.D</td>
<td>The database directory <code>&lt;directoryName&gt;</code> exists. However, it does not contain the expected <code>&lt;servicePropertiesName&gt;</code> 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.</td>
</tr>
<tr>
<td>XBM0B.D</td>
<td>Failed to edit/write service properties file: <code>&lt;errorMessage&gt;</code></td>
</tr>
<tr>
<td>XBM0C.D</td>
<td>Missing privilege for operation <code>&lt;operation&gt;</code> on file <code>&lt;path&gt;</code>: <code>&lt;errorMessage&gt;</code></td>
</tr>
<tr>
<td>XBM0G.D</td>
<td>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.</td>
</tr>
<tr>
<td>XBM0H.D</td>
<td>Directory <code>&lt;directoryName&gt;</code> cannot be created.</td>
</tr>
<tr>
<td>XBM0I.D</td>
<td>Directory <code>&lt;directoryName&gt;</code> cannot be removed.</td>
</tr>
<tr>
<td>XBM0J.D</td>
<td>Directory <code>&lt;directoryName&gt;</code> already exists.</td>
</tr>
<tr>
<td>XBM0K.D</td>
<td>Unknown sub-protocol for database name <code>&lt;databaseName&gt;</code>.</td>
</tr>
<tr>
<td>XBM0L.D</td>
<td>Specified authentication scheme class <code>&lt;className&gt;</code> does implement the authentication interface <code>&lt;interfaceName&gt;</code></td>
</tr>
<tr>
<td>XBM0M.D</td>
<td>Error creating an instance of a class named <code>&lt;className&gt;</code>. 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: <code>&lt;detail&gt;</code></td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>XBM0N.D</td>
<td>JDBC Driver registration with java.sql.DriverManager failed. See next exception for details.</td>
</tr>
<tr>
<td>XBM0P.D</td>
<td>Service provider is read-only. Operation not permitted.</td>
</tr>
<tr>
<td>XBM0Q.D</td>
<td>File <code>&lt;fileName&gt;</code> not found. Please make sure that backup copy is the correct one and it is not corrupted.</td>
</tr>
<tr>
<td>XBM0R.D</td>
<td>Unable to remove File <code>&lt;fileName&gt;</code>.</td>
</tr>
<tr>
<td>XBM0S.D</td>
<td>Unable to rename file <code>'&lt;fileName&gt;'</code> to <code>'&lt;fileName&gt;'</code>.</td>
</tr>
<tr>
<td>XBM0T.D</td>
<td>Ambiguous sub-protocol for database name <code>&lt;databaseName&gt;</code>.</td>
</tr>
<tr>
<td>XBM0U.S</td>
<td>No class was registered for identifier <code>&lt;identifierName&gt;</code>.</td>
</tr>
<tr>
<td>XBM0V.S</td>
<td>An exception was thrown while loading class <code>&lt;className&gt;</code> registered for identifier <code>&lt;identifierName&gt;</code>.</td>
</tr>
<tr>
<td>XBM0W.S</td>
<td>An exception was thrown while creating an instance of class <code>&lt;className3&gt;</code> registered for identifier <code>&lt;identifierName&gt;</code>.</td>
</tr>
<tr>
<td>XBM0X.D</td>
<td>Supplied territory description <code>'&lt;value&gt;'</code> is invalid, expecting ln[<em>.CO[</em>.variant]] ln=lower-case two-letter ISO-639 language code, CO=upper-case two-letter ISO-3166 country codes, see java.util.Locale.</td>
</tr>
<tr>
<td>XBM03.D</td>
<td>Supplied value <code>'&lt;value&gt;'</code> for collation attribute is invalid, expecting UCS_BASIC or TERRITORY_BASED.</td>
</tr>
<tr>
<td>XBM04.D</td>
<td>Collator support not available from the JVM for the database's locale <code>'&lt;value&gt;'</code>.</td>
</tr>
<tr>
<td>XBM0Y.D</td>
<td>Backup database directory <code>&lt;directoryName&gt;</code> not found. Please make sure that the specified backup path is right.</td>
</tr>
<tr>
<td>XBM0Z.D</td>
<td>Unable to copy file <code>'&lt;fileName&gt;'</code> to <code>'&lt;fileName&gt;'</code>. Please make sure that there is enough space and permissions are correct.</td>
</tr>
</tbody>
</table>
## Error Class XCL: Execution exceptions

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCL01.S</td>
<td>Result set does not return rows. Operation <code>&lt;operationName&gt;</code> not permitted.</td>
</tr>
<tr>
<td>XCL05.S</td>
<td>Activation closed, operation <code>&lt;operationName&gt;</code> not permitted.</td>
</tr>
<tr>
<td>XCL07.S</td>
<td>Cursor <code>'&lt;cursorName&gt;'</code> is closed. Verify that autocommit is OFF.</td>
</tr>
<tr>
<td>XCL08.S</td>
<td>Cursor <code>'&lt;cursorName&gt;'</code> is not on a row.</td>
</tr>
<tr>
<td>XCL09.S</td>
<td>An Activation was passed to the <code>'&lt;methodName&gt;'</code> method that does not match the PreparedStatement.</td>
</tr>
<tr>
<td>XCL10.S</td>
<td>A PreparedStatement has been recompiled and the parameters have changed. If you are using JDBC you must prepare the statement again.</td>
</tr>
<tr>
<td>XCL12.S</td>
<td>An attempt was made to put a data value of type <code>'&lt;datatypeName&gt;'</code> into a data value of type <code>'&lt;datatypeName&gt;'</code>.</td>
</tr>
<tr>
<td>XCL13.S</td>
<td>The parameter position <code>'&lt;parameterPosition&gt;'</code> is out of range. The number of parameters for this prepared statement is <code>'&lt;number&gt;'</code>.</td>
</tr>
<tr>
<td>XCL14.S</td>
<td>The column position <code>'&lt;columnPosition&gt;'</code> is out of range. The number of columns for this ResultSet is <code>'&lt;number&gt;'</code>.</td>
</tr>
<tr>
<td>XCL15.S</td>
<td>A ClassCastException occurred when calling the compareTo() method on an object <code>'&lt;object&gt;'</code>. The parameter to compareTo() is of class <code>'&lt;className&gt;'</code>.</td>
</tr>
<tr>
<td>XCL16.S</td>
<td>ResultSet not open. Operation <code>&lt;operation&gt;</code> not permitted. Verify that autocommit is OFF.</td>
</tr>
<tr>
<td>XCL18.S</td>
<td>Stream or LOB value cannot be retrieved more than once</td>
</tr>
<tr>
<td>XCL19.S</td>
<td>Missing row in table <code>'&lt;tableName&gt;'</code> for key <code>'&lt;key&gt;'</code>.</td>
</tr>
<tr>
<td>XCL20.S</td>
<td>Catalogs at version level <code>'&lt;versionNumber&gt;'</code> cannot be upgraded to version level <code>'&lt;versionNumber&gt;'</code>.</td>
</tr>
<tr>
<td>XCL21.S</td>
<td>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.</td>
</tr>
<tr>
<td>XCL22.S</td>
<td>Parameter <code>&lt;parameterName&gt;</code> cannot be registered as an OUT parameter because it is an IN parameter.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>XCL23.S</td>
<td>SQL type number '&lt;type&gt;' is not a supported type by registerOutParameter().</td>
</tr>
<tr>
<td>XCL24.S</td>
<td>Parameter '&lt;parameterName&gt;' 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 '&lt;type&gt;'.</td>
</tr>
<tr>
<td>XCL25.S</td>
<td>Parameter '&lt;parameterName&gt;' cannot be registered to be of type '&lt;type&gt;' because it maps to type '&lt;type&gt;' and they are incompatible.</td>
</tr>
<tr>
<td>XCL26.S</td>
<td>Parameter '&lt;parameterName&gt;' is not an output parameter.</td>
</tr>
<tr>
<td>XCL27.S</td>
<td>Return output parameters cannot be set.</td>
</tr>
<tr>
<td>XCL30.S</td>
<td>An IOException was thrown when reading a '&lt;value&gt;' from an InputStream.</td>
</tr>
<tr>
<td>XCL31.S</td>
<td>Statement closed.</td>
</tr>
<tr>
<td>XCL33.S</td>
<td>The table cannot be defined as a dependent of table '&lt;tableName&gt;' because of delete rule restrictions. (The relationship is self-referencing and a self-referencing relationship already exists with the SET NULL delete rule.)</td>
</tr>
<tr>
<td>XCL34.S</td>
<td>The table cannot be defined as a dependent of table '&lt;tableName&gt;' 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)).</td>
</tr>
<tr>
<td>XCL35.S</td>
<td>The table cannot be defined as a dependent of table '&lt;tableName&gt;' 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.).</td>
</tr>
<tr>
<td>XCL36.S</td>
<td>The delete rule of foreign key must be '&lt;value&gt;'. (The referential constraint is self-referencing and an existing self-referencing constraint has the indicated delete rule (NO ACTION, RESTRICT or CASCADE).)</td>
</tr>
<tr>
<td>XCL37.S</td>
<td>The delete rule of foreign key must be '&lt;value&gt;'. (The referential constraint is self-referencing and the table is dependent in a relationship with a delete rule of CASCADE.)</td>
</tr>
<tr>
<td>XCL38.S</td>
<td>The delete rule of foreign key must be '&lt;ruleName&gt;'. (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).)</td>
</tr>
<tr>
<td>XCL39.S</td>
<td>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.)</td>
</tr>
<tr>
<td>XCL40.S</td>
<td>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.)</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>XCL41.S</td>
<td>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.)</td>
</tr>
<tr>
<td>XCL42.S</td>
<td>CASCADE</td>
</tr>
<tr>
<td>XCL43.S</td>
<td>SET NULL</td>
</tr>
<tr>
<td>XCL44.S</td>
<td>RESTRICT</td>
</tr>
<tr>
<td>XCL45.S</td>
<td>NO ACTION</td>
</tr>
<tr>
<td>XCL46.S</td>
<td>SET DEFAULT</td>
</tr>
<tr>
<td>XCL47.S</td>
<td>Use of '&lt;value&gt;' requires database to be upgraded from version &lt;versionNumber&gt; to version &lt;versionNumber&gt; or later.</td>
</tr>
<tr>
<td>XCL48.S</td>
<td>TRUNCATE TABLE is not permitted on '&lt;value&gt;' because unique/primary key constraints on this table are referenced by enabled foreign key constraints from other tables.</td>
</tr>
<tr>
<td>XCL49.S</td>
<td>TRUNCATE TABLE is not permitted on '&lt;value&gt;' because it has an enabled DELETE trigger '&lt;value&gt;'.</td>
</tr>
<tr>
<td>XCL50.S</td>
<td>Upgrading the database from a previous version is not supported. The database being accessed is at version level '&lt;versionNumber&gt;', this software is at version level '&lt;versionNumber&gt;'.</td>
</tr>
<tr>
<td>XCL51.S</td>
<td>The requested function can not reference tables in SESSION schema.</td>
</tr>
<tr>
<td>XCL52.S</td>
<td>The statement has been cancelled or timed out.</td>
</tr>
</tbody>
</table>
## Error Class XCW: Upgrade unsupported

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCW00.D</td>
<td>Unsupported upgrade from '&lt;value&gt;' to '&lt;value&gt;'.</td>
</tr>
</tbody>
</table>
## Error Class XCX: Internal Utility Errors

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCXA0.S</td>
<td>Invalid identifier: '&lt;value&gt;'.</td>
</tr>
<tr>
<td>XCXB0.S</td>
<td>Invalid database classpath: '&lt;classpath&gt;'.</td>
</tr>
<tr>
<td>XCXC0.S</td>
<td>Invalid id list.</td>
</tr>
<tr>
<td>XCXE0.S</td>
<td>You are trying to do an operation that uses the territory of the database, but the database does not have a territory.</td>
</tr>
</tbody>
</table>
## Error Class XCY: Splice Property Exceptions

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCY00.S</td>
<td>Invalid value for property '&lt;value&gt;'='&lt;value&gt;'.</td>
</tr>
<tr>
<td>XCY02.S</td>
<td>The requested property change is not supported '&lt;value&gt;'='&lt;value&gt;'.</td>
</tr>
<tr>
<td>XCY03.S</td>
<td>Required property '&lt;propertyName&gt;' has not been set.</td>
</tr>
<tr>
<td>XCY04.S</td>
<td>Invalid syntax for optimizer overrides. The syntax should be -- SPLICE-PROPERTIES propertyName = value [, propertyName = value]*</td>
</tr>
<tr>
<td>XCY05.S.2</td>
<td>Invalid setting of the derby.authentication.provider property. This property is already set to enable NATIVE authentication and cannot be changed.</td>
</tr>
<tr>
<td>XCY05.S.3</td>
<td>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.</td>
</tr>
</tbody>
</table>
## Error Class XCZ: com.splicemachine.db.database.UserUtility

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCZ00.S</td>
<td>Unknown permission '&lt;permissionName&gt;'.</td>
</tr>
<tr>
<td>XCZ01.S</td>
<td>Unknown user '&lt;authorizationName&gt;'.</td>
</tr>
<tr>
<td>XCZ02.S</td>
<td>Invalid parameter '&lt;value&gt;'='&lt;value&gt;'.</td>
</tr>
</tbody>
</table>
## Error Class XD00: Dependency Manager

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XD003.S</td>
<td>Unable to restore dependency from disk. DependableFinder = '&lt;value&gt;'. Further information: '&lt;value&gt;'.</td>
</tr>
<tr>
<td>XD004.S</td>
<td>Unable to store dependencies.</td>
</tr>
</tbody>
</table>
## Error Class XIE: Import/Export Exceptions

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XIE01.S</td>
<td>Connection was null.</td>
</tr>
<tr>
<td>XIE03.S</td>
<td>Data found on line &lt;lineNumber&gt; for column &lt;columnName&gt; after the stop delimiter.</td>
</tr>
<tr>
<td>XIE04.S</td>
<td>Data file not found: &lt;fileName&gt;</td>
</tr>
<tr>
<td>XIE05.S</td>
<td>Data file cannot be null.</td>
</tr>
<tr>
<td>XIE06.S</td>
<td>Entity name was null.</td>
</tr>
<tr>
<td>XIE07.S</td>
<td>Field and record separators cannot be substrings of each other.</td>
</tr>
<tr>
<td>XIE08.S</td>
<td>There is no column named: &lt;columnName&gt;.</td>
</tr>
<tr>
<td>XIE09.S</td>
<td>The total number of columns in the row is: &lt;number&gt;.</td>
</tr>
<tr>
<td>XIE0A.S</td>
<td>Number of columns in column definition, &lt;columnName&gt;, differ from those found in import file &lt;type&gt;.</td>
</tr>
<tr>
<td>XIE0B.S</td>
<td>Column '&lt;columnName&gt;' in the table is of type &lt;type&gt;, it is not supported by the import/export feature.</td>
</tr>
<tr>
<td>XIE0C.S</td>
<td>Illegal &lt;delimiter&gt; delimiter character '&lt;character&gt;'.</td>
</tr>
<tr>
<td>XIE0D.S</td>
<td>Cannot find the record separator on line &lt;lineNumber&gt;.</td>
</tr>
<tr>
<td>XIE0E.S</td>
<td>Read endOfFile at unexpected place on line &lt;lineNumber&gt;.</td>
</tr>
<tr>
<td>XIE0F.S</td>
<td>Character delimiter cannot be the same as the column delimiter.</td>
</tr>
<tr>
<td>XIE0I.S</td>
<td>An IOException occurred while writing data to the file.</td>
</tr>
<tr>
<td>XIE0J.S</td>
<td>A delimiter is not valid or is used more than once.</td>
</tr>
<tr>
<td>XIE0K.S</td>
<td>The period was specified as a character string delimiter.</td>
</tr>
<tr>
<td>XIE0M.S</td>
<td>Table '&lt;tableName&gt;' does not exist.</td>
</tr>
<tr>
<td>XIE0N.S</td>
<td>An invalid hexadecimal string '&lt;hexString&gt;' detected in the import file.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>XIE0P.S</td>
<td>Lob data file <code>&lt;fileName&gt;</code> referenced in the import file not found.</td>
</tr>
<tr>
<td>XIE0Q.S</td>
<td>Lob data file name cannot be null.</td>
</tr>
<tr>
<td>XIE0R.S</td>
<td>Import error on line <code>&lt;lineNumber&gt;</code> of file <code>&lt;fileName&gt;</code>: <code>&lt;details&gt;</code></td>
</tr>
<tr>
<td>XIE10.S</td>
<td>Import error during reading source file <code>&lt;fileName&gt;</code>: <code>&lt;details&gt;</code></td>
</tr>
<tr>
<td>XIE11.S</td>
<td>SuperCSVReader error during Import: <code>&lt;details&gt;</code></td>
</tr>
<tr>
<td>XIE12.S</td>
<td>There was <code>&lt;details&gt;</code> RegionServer failures during a write with WAL disabled, the transaction has to rollback to avoid data loss.</td>
</tr>
<tr>
<td>XIE0S.S</td>
<td>The export operation was not performed, because the specified output file <code>&lt;fileName&gt;</code> 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.</td>
</tr>
<tr>
<td>XIE0T.S</td>
<td>The export operation was not performed, because the specified large object auxiliary file <code>&lt;fileName&gt;</code> 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.</td>
</tr>
<tr>
<td>XIE0U.S</td>
<td>The export operation was not performed, because the specified parameter (replicationCount) is less than or equal to zero.</td>
</tr>
<tr>
<td>XIE0X.S</td>
<td>The export operation was not performed, because value of the specified parameter <code>&lt;paramName&gt;</code> is wrong.</td>
</tr>
</tbody>
</table>
## Error Class XJ: Connectivity Errors

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XJ004.C</td>
<td>Database '&lt;databaseName&gt;' not found.</td>
</tr>
<tr>
<td>XJ008.S</td>
<td>Cannot rollback or release a savepoint when in auto-commit mode.</td>
</tr>
<tr>
<td>XJ009.S</td>
<td>Use of CallableStatement required for stored procedure call or use of output parameters: &lt;value&gt;</td>
</tr>
<tr>
<td>XJ010.S</td>
<td>Cannot issue savepoint when autoCommit is on.</td>
</tr>
<tr>
<td>XJ011.S</td>
<td>Cannot pass null for savepoint name.</td>
</tr>
<tr>
<td>XJ012.S</td>
<td>'&lt;value&gt;' already closed.</td>
</tr>
<tr>
<td>XJ013.S</td>
<td>No ID for named savepoints.</td>
</tr>
<tr>
<td>XJ014.S</td>
<td>No name for un-named savepoints.</td>
</tr>
<tr>
<td>XJ015.M</td>
<td>Splice system shutdown.</td>
</tr>
<tr>
<td>XJ016.S</td>
<td>Method '&lt;methodName&gt;' not allowed on prepared statement.</td>
</tr>
<tr>
<td>XJ017.S</td>
<td>No savepoint command allowed inside the trigger code.</td>
</tr>
<tr>
<td>XJ018.S</td>
<td>Column name cannot be null.</td>
</tr>
<tr>
<td>XJ020.S</td>
<td>Object type not convertible to TYPE '&lt;typeName&gt;', invalid java.sql.Types value, or object was null.</td>
</tr>
<tr>
<td>XJ021.S</td>
<td>Type is not supported.</td>
</tr>
<tr>
<td>XJ022.S</td>
<td>Unable to set stream: '&lt;name&gt;'.</td>
</tr>
<tr>
<td>XJ023.S</td>
<td>Input stream did not have exact amount of data as the requested length.</td>
</tr>
<tr>
<td>XJ025.S</td>
<td>Input stream cannot have negative length.</td>
</tr>
<tr>
<td>XJ028.C</td>
<td>The URL '&lt;urlValue&gt;' is not properly formed.</td>
</tr>
<tr>
<td>XJ030.S</td>
<td>Cannot set AUTOCOMMIT ON when in a nested connection.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XJ040.C</td>
<td>Failed to start database '&lt;databaseName&gt;' with class loader '&lt;classLoader&gt;', see the next exception for details.</td>
</tr>
<tr>
<td>XJ041.C</td>
<td>Failed to create database '&lt;databaseName&gt;', see the next exception for details.</td>
</tr>
<tr>
<td>XJ042.S</td>
<td>'&lt;value&gt;' is not a valid value for property '&lt;propertyName&gt;'.</td>
</tr>
<tr>
<td>XJ044.S</td>
<td>'&lt;value&gt;' is an invalid scale.</td>
</tr>
<tr>
<td>XJ045.S</td>
<td>Invalid or (currently) unsupported isolation level, '&lt;levelName&gt;', passed to Connection.setTransactionIsolation(). The currently supported</td>
</tr>
<tr>
<td></td>
<td>values are java.sql.Connection.TRANSACTION_SERIALIZABLE, java.sql.Connection.TRANSACTION_REPEATABLE_READ, java.sql.Connection.TRANSACTION_</td>
</tr>
<tr>
<td></td>
<td>READ_COMMITTED, and java.sql.Connection.TRANSACTION_READ_UNCOMMITTED.</td>
</tr>
<tr>
<td>XJ048.C</td>
<td>Conflicting boot attributes specified: '&lt;attributes&gt;'</td>
</tr>
<tr>
<td>XJ049.C</td>
<td>Conflicting create attributes specified.</td>
</tr>
<tr>
<td>XJ04B.S</td>
<td>Batch cannot contain a command that attempts to return a result set.</td>
</tr>
<tr>
<td>XJ04C.S</td>
<td>CallableStatement batch cannot contain output parameters.</td>
</tr>
<tr>
<td>XJ056.S</td>
<td>Cannot set AUTOCOMMIT ON when in an XA connection.</td>
</tr>
<tr>
<td>XJ057.S</td>
<td>Cannot commit a global transaction using the Connection, commit processing must go thru XAResource interface.</td>
</tr>
<tr>
<td>XJ058.S</td>
<td>Cannot rollback a global transaction using the Connection, commit processing must go thru XAResource interface.</td>
</tr>
<tr>
<td>XJ059.S</td>
<td>Cannot close a connection while a global transaction is still active.</td>
</tr>
<tr>
<td>XJ05B.C</td>
<td>JDBC attribute '&lt;attributeName&gt;' has an invalid value '&lt;value&gt;', valid values are '&lt;value&gt;'.</td>
</tr>
<tr>
<td>XJ05C.S</td>
<td>Cannot set holdability ResultSet.HOLD_CURSORS_OVER_COMMIT for a global transaction.</td>
</tr>
<tr>
<td>XJ061.S</td>
<td>The '&lt;methodName&gt;' method is only allowed on scroll cursors.</td>
</tr>
<tr>
<td>XJ062.S</td>
<td>Invalid parameter value '&lt;value&gt;' for ResultSet.setFetchSize(int rows).</td>
</tr>
<tr>
<td>XJ063.S</td>
<td>Invalid parameter value '&lt;value&gt;' for Statement.setMaxRows(int maxRows). Parameter value must be &gt;= 0.</td>
</tr>
<tr>
<td>XJ064.S</td>
<td>Invalid parameter value '&lt;value&gt;' for setFetchDirection(int direction).</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>XJ065.S</td>
<td>Invalid parameter value 'value' for Statement.setFetchSize(int rows).</td>
</tr>
<tr>
<td>XJ066.S</td>
<td>Invalid parameter value 'value' for Statement.setMaxFieldSize(int max).</td>
</tr>
<tr>
<td>XJ067.S</td>
<td>SQL text pointer is null.</td>
</tr>
<tr>
<td>XJ068.S</td>
<td>Only executeBatch and clearBatch allowed in the middle of a batch.</td>
</tr>
<tr>
<td>XJ069.S</td>
<td>No SetXXX methods allowed in case of USING execute statement.</td>
</tr>
<tr>
<td>XJ070.S</td>
<td>Negative or zero position argument 'value' passed in a Blob or Clob method.</td>
</tr>
<tr>
<td>XJ071.S</td>
<td>Negative length argument 'value' passed in a BLOB or CLOB method.</td>
</tr>
<tr>
<td>XJ072.S</td>
<td>Null pattern or searchStr passed in to a BLOB or CLOB position method.</td>
</tr>
<tr>
<td>XJ073.S</td>
<td>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.</td>
</tr>
<tr>
<td>XJ074.S</td>
<td>Invalid parameter value 'value' for Statement.setQueryTimeout(int seconds).</td>
</tr>
<tr>
<td>XJ076.S</td>
<td>The position argument 'value' exceeds the size of the BLOB/CLOB.</td>
</tr>
<tr>
<td>XJ077.S</td>
<td>Got an exception when trying to read the first byte/character of the BLOB/CLOB pattern using getBytes/getSubString.</td>
</tr>
<tr>
<td>XJ078.S</td>
<td>Offset 'value' is either less than zero or is too large for the current BLOB/CLOB.</td>
</tr>
<tr>
<td>XJ079.S</td>
<td>The length specified 'value' exceeds the size of the BLOB/CLOB.</td>
</tr>
<tr>
<td>XJ080.S</td>
<td>USING execute statement passed &lt;number&gt; parameters rather than &lt;number&gt;.</td>
</tr>
<tr>
<td>XJ081.C</td>
<td>Conflicting create/restore/recovery attributes specified.</td>
</tr>
<tr>
<td>XJ081.S</td>
<td>Invalid value 'value' passed as parameter 'parameterName' to method 'methodName'.</td>
</tr>
<tr>
<td>XJ085.S</td>
<td>Stream has already been read and end-of-file reached and cannot be re-used.</td>
</tr>
<tr>
<td>XJ086.S</td>
<td>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.</td>
</tr>
<tr>
<td>XJ087.S</td>
<td>Sum of position('pos') and length('length') is greater than the size of the LOB plus one.</td>
</tr>
<tr>
<td>XJ088.S</td>
<td>Invalid operation: wasNull() called with no data retrieved.</td>
</tr>
<tr>
<td>XJ090.S</td>
<td>Invalid parameter: calendar is null.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XJ091.S</td>
<td>Invalid argument: parameter index <code>&lt;indexNumber&gt;</code> is not an OUT or INOUT parameter.</td>
</tr>
<tr>
<td>XJ093.S</td>
<td>Length of BLOB/CLOB, <code>&lt;number&gt;</code>, is too large. The length cannot exceed <code>&lt;number&gt;</code>.</td>
</tr>
<tr>
<td>XJ095.S</td>
<td>An attempt to execute a privileged action failed.</td>
</tr>
<tr>
<td>XJ096.S</td>
<td>A resource bundle could not be found in the <code>&lt;packageName&gt;</code> package for <code>&lt;value&gt;</code></td>
</tr>
<tr>
<td>XJ097.S</td>
<td>Cannot rollback or release a savepoint that was not created by this connection.</td>
</tr>
<tr>
<td>XJ098.S</td>
<td>The auto-generated keys value <code>&lt;value&gt;</code> is invalid</td>
</tr>
<tr>
<td>XJ099.S</td>
<td>The Reader/Stream object does not contain length characters</td>
</tr>
<tr>
<td>XJ100.S</td>
<td>The scale supplied by the registerOutParameter method does not match with the setter method. Possible loss of precision!</td>
</tr>
<tr>
<td>XJ103.S</td>
<td>Table name can not be null</td>
</tr>
<tr>
<td>XJ104.S</td>
<td>Shared key length is invalid: <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XJ105.S</td>
<td>DES key has the wrong length, expected length <code>&lt;number&gt;</code>, got length <code>&lt;number&gt;</code>.</td>
</tr>
<tr>
<td>XJ106.S</td>
<td>No such padding</td>
</tr>
<tr>
<td>XJ107.S</td>
<td>Bad Padding</td>
</tr>
<tr>
<td>XJ108.S</td>
<td>Illegal Block Size</td>
</tr>
<tr>
<td>XJ110.S</td>
<td>Primary table name can not be null</td>
</tr>
<tr>
<td>XJ111.S</td>
<td>Foreign table name can not be null</td>
</tr>
<tr>
<td>XJ112.S</td>
<td>Security exception encountered, see next exception for details.</td>
</tr>
<tr>
<td>XJ113.S</td>
<td>Unable to open file <code>&lt;fileName&gt;</code> : <code>&lt;error&gt;</code></td>
</tr>
<tr>
<td>XJ114.S</td>
<td>Invalid cursor name <code>'&lt;cursorName&gt;'</code></td>
</tr>
<tr>
<td>XJ115.S</td>
<td>Unable to open resultSet with requested holdability <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XJ116.S</td>
<td>No more than <code>&lt;number&gt;</code> commands may be added to a single batch.</td>
</tr>
<tr>
<td>XJ117.S</td>
<td>Batching of queries not allowed by J2EE compliance.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>XJ118.S</td>
<td>Query batch requested on a non-query statement.</td>
</tr>
<tr>
<td>XJ121.S</td>
<td>Invalid operation at current cursor position.</td>
</tr>
<tr>
<td>XJ122.S</td>
<td>No updateXXX methods were called on this row.</td>
</tr>
<tr>
<td>XJ123.S</td>
<td>This method must be called to update values in the current row or the insert row.</td>
</tr>
<tr>
<td>XJ124.S</td>
<td>Column not updatable.</td>
</tr>
<tr>
<td>XJ125.S</td>
<td>This method should only be called on ResultSet objects that are scrollable (type TYPE_SCROLL_INSENSITIVE).</td>
</tr>
<tr>
<td>XJ126.S</td>
<td>This method should not be called on sensitive dynamic cursors.</td>
</tr>
<tr>
<td>XJ128.S</td>
<td>Unable to unwrap for '&lt;value&gt;'</td>
</tr>
<tr>
<td>XJ200.S</td>
<td>Exceeded maximum number of sections &lt;value&gt;</td>
</tr>
<tr>
<td>XJ202.S</td>
<td>Invalid cursor name '&lt;cursorName&gt;'.</td>
</tr>
<tr>
<td>XJ203.S</td>
<td>Cursor name '&lt;cursorName&gt;' is already in use</td>
</tr>
<tr>
<td>XJ204.S</td>
<td>Unable to open result set with requested holdability &lt;holdValue&gt;.</td>
</tr>
<tr>
<td>XJ206.S</td>
<td>SQL text '&lt;value&gt;' has no tokens.</td>
</tr>
<tr>
<td>XJ207.S</td>
<td>executeQuery method can not be used for update.</td>
</tr>
<tr>
<td>XJ208.S</td>
<td>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.</td>
</tr>
<tr>
<td>XJ209.S</td>
<td>The required stored procedure is not installed on the server.</td>
</tr>
<tr>
<td>XJ210.S</td>
<td>The load module name for the stored procedure on the server is not found.</td>
</tr>
<tr>
<td>XJ211.S</td>
<td>Non-recoverable chain-breaking exception occurred during batch processing. The batch is terminated non-atomically.</td>
</tr>
<tr>
<td>XJ212.S</td>
<td>Invalid attribute syntax: &lt;attributeSyntax&gt;</td>
</tr>
<tr>
<td>XJ213.C</td>
<td>The traceLevel connection property does not have a valid format for a number.</td>
</tr>
<tr>
<td>XJ214.S</td>
<td>An IO Error occurred when calling free() on a CLOB or BLOB.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
</tr>
<tr>
<td>XJ215.S</td>
<td>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.</td>
</tr>
<tr>
<td>XJ216.S</td>
<td>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.</td>
</tr>
<tr>
<td>XJ217.S</td>
<td>The locator that was supplied for this CLOB/BLOB is invalid</td>
</tr>
</tbody>
</table>
### Error Class XK: Security Exceptions

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XK000.S</td>
<td>The security policy could not be reloaded: &lt;reason&gt;</td>
</tr>
<tr>
<td>XK001.S</td>
<td>Username not found in SYS.SYSUSERS.</td>
</tr>
</tbody>
</table>
## Error Class XN: Network Client Exceptions

### Error Class XN: Network Client Exceptions

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XN001.S</td>
<td>Connection reset is not allowed when inside a unit of work.</td>
</tr>
<tr>
<td>XN008.S</td>
<td>Query processing has been terminated due to an error on the server.</td>
</tr>
<tr>
<td>XN009.S</td>
<td>Error obtaining length of BLOB/CLOB object, exception follows.</td>
</tr>
<tr>
<td>XN010.S</td>
<td>Procedure name can not be null.</td>
</tr>
<tr>
<td>XN011.S</td>
<td>Procedure name length &lt;number&gt; is not within the valid range of 1 to &lt;number&gt;.</td>
</tr>
<tr>
<td>XN012.S</td>
<td>On &lt;operatingSystemName&gt; platforms, XA supports version &lt;versionNumber&gt; and above, this is version &lt;versionNumber&gt;</td>
</tr>
<tr>
<td>XN013.S</td>
<td>Invalid scroll orientation.</td>
</tr>
<tr>
<td>XN014.S</td>
<td>Encountered an Exception while reading from the stream specified by parameter #&lt;value&gt;. The remaining data expected by the server has been filled with 0x0. The Exception had this message: &lt;messageText&gt;.</td>
</tr>
<tr>
<td>XN015.S</td>
<td>Network protocol error: the specified size of the InputStream, parameter #&lt;value&gt;, is less than the actual InputStream length.</td>
</tr>
<tr>
<td>XN016.S</td>
<td>Encountered an Exception while trying to verify the length of the stream specified by parameter #&lt;value&gt;. The Exception had this message: &lt;messageText&gt;.</td>
</tr>
<tr>
<td>XN017.S</td>
<td>End of stream prematurely reached while reading the stream specified by parameter #&lt;value&gt;. The remaining data expected by the server has been filled with 0x0.</td>
</tr>
<tr>
<td>XN018.S</td>
<td>Network protocol error: the specified size of the Reader, parameter #&lt;value&gt;, is less than the actual InputStream length.</td>
</tr>
<tr>
<td>XN019.S</td>
<td>Error executing a &lt;value&gt;, server returned &lt;value&gt;.</td>
</tr>
<tr>
<td>XN020.S</td>
<td>Error marshalling or unmarshalling a user defined type: &lt;messageDetail&gt;</td>
</tr>
<tr>
<td>XN021.S</td>
<td>An object of type &lt;sourceClassName&gt; cannot be cast to an object of type &lt;targetClassName&gt;.</td>
</tr>
</tbody>
</table>
## Error Class XRE: Replication Exceptions

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XRE00</td>
<td>This LogFactory module does not support replication.</td>
</tr>
<tr>
<td>XRE01</td>
<td>The log received from the master is corrupted.</td>
</tr>
<tr>
<td>XRE02</td>
<td>Master and Slave at different versions. Unable to proceed with Replication.</td>
</tr>
<tr>
<td>XRE03</td>
<td>Unexpected replication error. See derby.log for details.</td>
</tr>
<tr>
<td>XRE04.C.1</td>
<td>Could not establish a connection to the peer of the replicated database '&lt;dbname&gt;' on address '&lt;hostname&gt;':&lt;portname&gt;'.</td>
</tr>
<tr>
<td>XRE04.C.2</td>
<td>Connection lost for replicated database '&lt;dbname&gt;'.</td>
</tr>
<tr>
<td>XRE05.C</td>
<td>The log files on the master and slave are not in synch for replicated database '&lt;dbname&gt;'. The master log instant is &lt;masterfile&gt;:&lt;masteroffset&gt;, whereas the slave log instant is &lt;slavefile&gt;:&lt;slaveoffset&gt;. This is FATAL for replication - replication will be stopped.</td>
</tr>
<tr>
<td>XRE06</td>
<td>The connection attempts to the replication slave for the database &lt;dbname&gt; exceeded the specified timeout period.</td>
</tr>
<tr>
<td>XRE07</td>
<td>Could not perform operation because the database is not in replication master mode.</td>
</tr>
<tr>
<td>XRE08</td>
<td>Replication slave mode started successfully for database '&lt;dbname&gt;'. Connection refused because the database is in replication slave mode.</td>
</tr>
<tr>
<td>XRE09.C</td>
<td>Cannot start replication slave mode for database '&lt;dbname&gt;'. The database has already been booted.</td>
</tr>
<tr>
<td>XRE10</td>
<td>Conflicting attributes specified. See reference manual for attributes allowed in combination with replication attribute '&lt;attribute&gt;'.</td>
</tr>
<tr>
<td>XRE11.C</td>
<td>Could not perform operation '&lt;command&gt;' because the database '&lt;dbname&gt;' has not been booted.</td>
</tr>
<tr>
<td>XRE12</td>
<td>Replication network protocol error for database '&lt;dbname&gt;'. Expected message type '&lt;expectedtype&gt;', but received type '&lt;expectedtype&gt;'.</td>
</tr>
<tr>
<td>XRE20.D</td>
<td>Failover performed successfully for database '&lt;dbname&gt;', the database has been shutdown.</td>
</tr>
<tr>
<td>XRE21.C</td>
<td>Error occurred while performing failover for database '&lt;dbname&gt;', Failover attempt was aborted.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XRE22.C</td>
<td>Replication master has already been booted for database '&lt;dbname&gt;'</td>
</tr>
<tr>
<td>XRE23</td>
<td>Replication master cannot be started since unlogged operations are in progress, unfreeze to allow unlogged operations to complete and restart replication</td>
</tr>
<tr>
<td>XRE40</td>
<td>Could not perform operation because the database is not in replication slave mode.</td>
</tr>
<tr>
<td>XRE41.C</td>
<td>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.</td>
</tr>
<tr>
<td>XRE42.C</td>
<td>Replicated database '&lt;dbname&gt;' shutdown.</td>
</tr>
<tr>
<td>XRE43</td>
<td>Unexpected error when trying to stop replication slave mode. To stop replication slave mode, use operation 'stopSlave' or 'failover'.</td>
</tr>
</tbody>
</table>
Error Class XSAI: Store - access.protocol.interface

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSAI2.S</td>
<td>The conglomerate (&lt;value&gt;) requested does not exist.</td>
</tr>
<tr>
<td>XSAI3.S</td>
<td>Feature not implemented.</td>
</tr>
</tbody>
</table>
**Error Class XSAM: Store - AccessManager**

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSAM0 . S</td>
<td>Exception encountered while trying to boot module for '&lt;value&gt;'.</td>
</tr>
<tr>
<td>XSAM2 . S</td>
<td>There is no index or conglomerate with conglom id '&lt;conglomID&gt;' to drop.</td>
</tr>
<tr>
<td>XSAM3 . S</td>
<td>There is no index or conglomerate with conglom id '&lt;conglomID&gt;'.</td>
</tr>
<tr>
<td>XSAM4 . S</td>
<td>There is no sort called '&lt;sortName&gt;'.</td>
</tr>
<tr>
<td>XSAM5 . S</td>
<td>Scan must be opened and positioned by calling next() before making other calls.</td>
</tr>
<tr>
<td>XSAM6 . S</td>
<td>Record &lt;recordNumber&gt; on page &lt;pageNumber&gt; in container &lt;containerName&gt; not found.</td>
</tr>
</tbody>
</table>
## Error Class XSAS: Store - Sort

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSAS0 . S</td>
<td>A scan controller interface method was called which is not appropriate for a scan on a sort.</td>
</tr>
<tr>
<td>XSAS1 . S</td>
<td>An attempt was made to fetch a row before the beginning of a sort or after the end of a sort.</td>
</tr>
<tr>
<td>XSAS3 . S</td>
<td>The type of a row inserted into a sort does not match the sort's template.</td>
</tr>
<tr>
<td>XSAS6 . S</td>
<td>Could not acquire resources for sort.</td>
</tr>
</tbody>
</table>
## Error Class XSAX: Store - access.protocol.XA statement

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSAX0.S</td>
<td>XA protocol violation.</td>
</tr>
<tr>
<td>XSAX1.S</td>
<td>An attempt was made to start a global transaction with an Xid of an existing global transaction.</td>
</tr>
</tbody>
</table>
## Error Class XSCB: Store - BTree

### Error Class XSCB: Store - BTree

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSCB0.S</td>
<td>Could not create container.</td>
</tr>
<tr>
<td>XSCB1.S</td>
<td>Container &lt;containerName&gt; not found.</td>
</tr>
<tr>
<td>XSCB2.S</td>
<td>The required property &lt;propertyName&gt; not found in the property list given to createConglomerate() for a btree secondary index.</td>
</tr>
<tr>
<td>XSCB3.S</td>
<td>Unimplemented feature.</td>
</tr>
<tr>
<td>XSCB4.S</td>
<td>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 (&lt;value&gt;).</td>
</tr>
<tr>
<td>XSCB5.S</td>
<td>During logical undo of a btree insert or delete the row could not be found in the tree.</td>
</tr>
<tr>
<td>XSCB6.S</td>
<td>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.</td>
</tr>
<tr>
<td>XSCB7.S</td>
<td>An internal error was encountered during a btree scan - current_rh is null = &lt;value&gt;, position key is null = &lt;value&gt;.</td>
</tr>
<tr>
<td>XSCB8.S</td>
<td>The btree conglomerate &lt;value&gt; is closed.</td>
</tr>
<tr>
<td>XSCB9.S</td>
<td>Reserved for testing.</td>
</tr>
</tbody>
</table>
## Error Class XSCG0: Conglomerate

### Error Class XSCG0: Conglomerate

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSCG0 . S</td>
<td>Could not create a template.</td>
</tr>
</tbody>
</table>
Error Class XSCH: Heap

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSCH0 .S</td>
<td>Could not create container.</td>
</tr>
<tr>
<td>XSCH1 .S</td>
<td>Container &lt;containerName&gt; not found.</td>
</tr>
<tr>
<td>XSCH4 .S</td>
<td>Conglomerate could not be created.</td>
</tr>
<tr>
<td>XSCH5 .S</td>
<td>In a base table there was a mismatch between the requested column number &lt;number&gt; and the maximum number of columns &lt;number&gt;.</td>
</tr>
<tr>
<td>XSCH6 .S</td>
<td>The heap container with container id &lt;containerID&gt; is closed.</td>
</tr>
<tr>
<td>XSCH7 .S</td>
<td>The scan is not positioned.</td>
</tr>
<tr>
<td>XSCH8 .S</td>
<td>The feature is not implemented.</td>
</tr>
</tbody>
</table>
Error Class XSDA: RawStore - Data.Generic statement

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSDA1.S</td>
<td>An attempt was made to access an out of range slot on a page</td>
</tr>
<tr>
<td>XSDA2.S</td>
<td>An attempt was made to update a deleted record</td>
</tr>
<tr>
<td>XSDA3.S</td>
<td>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.</td>
</tr>
<tr>
<td>XSDA4.S</td>
<td>An unexpected exception was thrown</td>
</tr>
<tr>
<td>XSDA5.S</td>
<td>An attempt was made to undelete a record that is not deleted</td>
</tr>
<tr>
<td>XSDA6.S</td>
<td>Column &lt;columnName&gt; of row is null, it needs to be set to point to an object.</td>
</tr>
<tr>
<td>XSDA7.S</td>
<td>Restore of a serializable or SQLData object of class &lt;className&gt;, attempted to read more data than was originally stored.</td>
</tr>
<tr>
<td>XSDA8.S</td>
<td>Exception during restore of a serializable or SQLData object of class &lt;className&gt;</td>
</tr>
<tr>
<td>XSDA9.S</td>
<td>Class not found during restore of a serializable or SQLData object of class &lt;className&gt;</td>
</tr>
<tr>
<td>XSDAA.S</td>
<td>Illegal time stamp &lt;value&gt;, either time stamp is from a different page or of incompatible implementation</td>
</tr>
<tr>
<td>XSDAB.S</td>
<td>cannot set a null time stamp</td>
</tr>
<tr>
<td>XSDAC.S</td>
<td>Attempt to move either rows or pages from one container to another.</td>
</tr>
<tr>
<td>XSDAD.S</td>
<td>Attempt to move zero rows from one page to another.</td>
</tr>
<tr>
<td>XSDAE.S</td>
<td>Can only make a record handle for special record handle id.</td>
</tr>
<tr>
<td>XSDAF.S</td>
<td>Using special record handle as if it were a normal record handle.</td>
</tr>
<tr>
<td>XSDAG.S</td>
<td>The allocation nested top transaction cannot open the container.</td>
</tr>
<tr>
<td>XSDAI.S</td>
<td>Page &lt;page&gt; being removed is already locked for deallocation.</td>
</tr>
<tr>
<td>XSDAJ.S</td>
<td>Exception during write of a serializable or SQLData object</td>
</tr>
<tr>
<td><strong>SQLSTATE</strong></td>
<td><strong>Message Text</strong></td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>XSDAK.S</td>
<td>Wrong page is gotten for record handle <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XSDAL.S</td>
<td>Record handle <code>&lt;value&gt;</code> unexpectedly points to overflow page.</td>
</tr>
<tr>
<td>XSDAM.S</td>
<td>Exception during restore of a SQLData object of class <code>&lt;className&gt;</code>. The specified class cannot be instantiated.</td>
</tr>
<tr>
<td>XSDAN.S</td>
<td>Exception during restore of a SQLData object of class <code>&lt;className&gt;</code>. The specified class encountered an illegal access exception.</td>
</tr>
<tr>
<td>XSDAO.S</td>
<td>Internal error: page <code>&lt;pageNumber&gt;</code> attempted latched twice.</td>
</tr>
</tbody>
</table>
Error Class XSDB: RawStore - Data.Generic transaction

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSDB0.D</td>
<td>Unexpected exception on in-memory page &lt;page&gt;</td>
</tr>
<tr>
<td>XSDB1.D</td>
<td>Unknown page format at page &lt;page&gt;</td>
</tr>
<tr>
<td>XSDB2.D</td>
<td>Unknown container format at container &lt;containerName&gt;: &lt;value&gt;</td>
</tr>
<tr>
<td>XSDB3.D</td>
<td>Container information cannot change once written: was &lt;value&gt;, now &lt;value&gt;</td>
</tr>
<tr>
<td>XSDB4.D</td>
<td>Page &lt;page&gt; is at version &lt;versionNumber&gt;, the log file contains change version &lt;versionNumber&gt;, either there are log records of this page missing, or this page did not get written out to disk properly.</td>
</tr>
<tr>
<td>XSDB5.D</td>
<td>Log has change record on page &lt;page&gt;, which is beyond the end of the container.</td>
</tr>
<tr>
<td>XSDB6.D</td>
<td>Another instance of Splice may have already booted the database &lt;databaseName&gt;.</td>
</tr>
<tr>
<td>XSDB7.D</td>
<td>WARNING: Splice (instance &lt;value&gt;) is attempting to boot the database &lt;databaseName&gt; even though Splice (instance &lt;value&gt;) 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.</td>
</tr>
<tr>
<td>XSDB8.D</td>
<td>WARNING: Splice (instance &lt;value&gt;) is attempting to boot the database &lt;databaseName&gt; even though Splice (instance &lt;value&gt;) may still be active. Only one instance of Splice should boot a database at a 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.</td>
</tr>
<tr>
<td>XSDB9.D</td>
<td>Stream container &lt;containerName&gt; is corrupt.</td>
</tr>
<tr>
<td>XSDBA.D</td>
<td>Attempt to allocate object &lt;object&gt; failed.</td>
</tr>
<tr>
<td>XSDBB.D</td>
<td>Unknown page format at page &lt;page&gt;, page dump follows: &lt;value&gt;</td>
</tr>
<tr>
<td>XSDBC.D</td>
<td>Write of container information to page 0 of container &lt;container&gt; failed. See nested error for more information.</td>
</tr>
</tbody>
</table>
## Error Class XSDF: RawStore - Data.Filesystem statement

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSDF0.S</td>
<td>Could not create file <code>&lt;fileName&gt;</code> as it already exists.</td>
</tr>
<tr>
<td>XSDF1.S</td>
<td>Exception during creation of file <code>&lt;fileName&gt;</code> for container</td>
</tr>
<tr>
<td>XSDF2.S</td>
<td>Exception during creation of file <code>&lt;fileName&gt;</code> for container, file could not be removed. The exception was: <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XSDF3.S</td>
<td>Cannot create segment <code>&lt;segmentName&gt;</code></td>
</tr>
<tr>
<td>XSDF4.S</td>
<td>Exception during remove of file <code>&lt;fileName&gt;</code> for dropped container, file could not be removed <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XSDF6.S</td>
<td>Cannot find the allocation page <code>&lt;page&gt;</code></td>
</tr>
<tr>
<td>XSDF7.S</td>
<td>Newly created page failed to be latched <code>&lt;value&gt;</code></td>
</tr>
<tr>
<td>XSDF8.S</td>
<td>Cannot find page <code>&lt;page&gt;</code> to reuse.</td>
</tr>
<tr>
<td>XSDFB.S</td>
<td>Operation not supported by a read only database</td>
</tr>
<tr>
<td>XSDFD.S</td>
<td>Different page image read on 2 I/Os on Page <code>&lt;page&gt;</code>, first image has incorrect checksum, second image has correct checksum. Page images follows: <code>&lt;value&gt;</code> <code>&lt;value&gt;</code></td>
</tr>
<tr>
<td>XSDFF.S</td>
<td>The requested operation failed due to an unexpected exception.</td>
</tr>
<tr>
<td>XSDFH.S</td>
<td>Cannot backup the database, got an I/O Exception while writing to the backup container file <code>&lt;fileName&gt;</code>.</td>
</tr>
<tr>
<td>XSDFI.S</td>
<td>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 <code>&lt;page&gt;</code>.</td>
</tr>
</tbody>
</table>
## Error Class XSDG: RawStore - Data.Filesystem database

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSDG0 . D</td>
<td>Page &lt;page&gt; could not be read from disk.</td>
</tr>
<tr>
<td>XSDG1 . D</td>
<td>Page &lt;page&gt; 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.</td>
</tr>
<tr>
<td>XSDG2 . D</td>
<td>Invalid checksum on Page &lt;page&gt;, expected=&lt;value&gt;, on-disk version=&lt;value&gt;, page dump follows: &lt;value&gt;</td>
</tr>
<tr>
<td>XSDG3 . D</td>
<td>Meta-data for &lt;containerName&gt; could not be accessed to &lt;type&gt; &lt;file&gt;</td>
</tr>
<tr>
<td>XSDG5 . D</td>
<td>Database is not in create mode when createFinished is called.</td>
</tr>
<tr>
<td>XSDG6 . D</td>
<td>Data segment directory not found in &lt;value&gt; backup during restore. Please make sure that backup copy is the right one and it is not corrupted.</td>
</tr>
<tr>
<td>XSDG7 . D</td>
<td>Directory &lt;directoryName&gt; could not be removed during restore. Please make sure that permissions are correct.</td>
</tr>
<tr>
<td>XSDG8 . D</td>
<td>Unable to copy directory '&lt;directoryName&gt;' to '&lt;value&gt;' during restore. Please make sure that there is enough space and permissions are correct.</td>
</tr>
<tr>
<td>XSDG9 . D</td>
<td>Splice thread received an interrupt during a disk I/O operation, please check your application for the source of the interrupt.</td>
</tr>
</tbody>
</table>
### Error Class XSLA: RawStore - Log.Generic database exceptions

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLA0.D</td>
<td>Cannot flush the log file to disk (&lt;value&gt;).</td>
</tr>
<tr>
<td>XSLA1.D</td>
<td>Log Record has been sent to the stream, but it cannot be applied to the store (Object (&lt;object&gt;)). This may cause recovery problems also.</td>
</tr>
<tr>
<td>XSLA2.D</td>
<td>System will shutdown, got I/O Exception while accessing log file.</td>
</tr>
<tr>
<td>XSLA3.D</td>
<td>Log Corrupted, has invalid data in the log stream.</td>
</tr>
<tr>
<td>XSLA4.D</td>
<td>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.</td>
</tr>
<tr>
<td>XSLA5.D</td>
<td>Cannot read log stream for some reason to rollback transaction (&lt;transactionID&gt;).</td>
</tr>
<tr>
<td>XSLA6.D</td>
<td>Cannot recover the database.</td>
</tr>
<tr>
<td>XSLA7.D</td>
<td>Cannot redo operation (&lt;operation&gt;) in the log.</td>
</tr>
<tr>
<td>XSLA8.D</td>
<td>Cannot rollback transaction (&lt;value&gt;), trying to compensate (&lt;value&gt;) operation with (&lt;value&gt;).</td>
</tr>
<tr>
<td>XSLAA.D</td>
<td>The store has been marked for shutdown by an earlier exception.</td>
</tr>
<tr>
<td>XSLAB.D</td>
<td>Cannot find log file (&lt;logfileName&gt;), please make sure your logDevice property is properly set with the correct path separator for your platform.</td>
</tr>
<tr>
<td>XSLAC.D</td>
<td>Database at (&lt;value&gt;) have incompatible format with the current version of software, it may have been created by or upgraded by a later version.</td>
</tr>
<tr>
<td>XSLAD.D</td>
<td>Log Record at instant (&lt;value&gt;) in log file (&lt;logfileName&gt;) corrupted. Expected log record length (&lt;value&gt;), real length (&lt;value&gt;).</td>
</tr>
<tr>
<td>XSLAE.D</td>
<td>Control file at (&lt;value&gt;) cannot be written or updated.</td>
</tr>
<tr>
<td>XSLAF.D</td>
<td>A Read Only database was created with dirty data buffers.</td>
</tr>
<tr>
<td>XSLAH.D</td>
<td>A Read Only database is being updated.</td>
</tr>
<tr>
<td>XSLAI.D</td>
<td>Cannot log the checkpoint log record.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XSLAJ.D</td>
<td>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.</td>
</tr>
<tr>
<td>XSLAK.D</td>
<td>Database has exceeded largest log file number <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XSLAL.D</td>
<td>log record size <code>&lt;value&gt;</code> exceeded the maximum allowable log file size <code>&lt;number&gt;</code>. Error encountered in log file <code>&lt;logfileName&gt;</code>, position <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XSLAM.D</td>
<td>Cannot verify database format at <code>&lt;value&gt;</code> due to IOException.</td>
</tr>
<tr>
<td>XSLAN.D</td>
<td>Database at <code>&lt;value&gt;</code> has an incompatible format with the current version of the software. The database was created by or upgraded by version <code>&lt;versionNumber&gt;</code>.</td>
</tr>
<tr>
<td>XSLAO.D</td>
<td>Recovery failed unexpected problem <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XSLAP.D</td>
<td>Database at <code>&lt;value&gt;</code> is at version <code>&lt;versionNumber&gt;</code>. Beta databases cannot be upgraded,</td>
</tr>
<tr>
<td>XSLAQ.D</td>
<td>cannot create log file at directory <code>&lt;directoryName&gt;</code>.</td>
</tr>
<tr>
<td>XSLAR.D</td>
<td>Unable to copy log file <code>&lt;logfileName&gt;</code> to <code>&lt;value&gt;</code> during restore. Please make sure that there is enough space and permissions are correct.</td>
</tr>
<tr>
<td>XSLAS.D</td>
<td>Log directory <code>&lt;directoryName&gt;</code> not found in backup during restore. Please make sure that backup copy is the correct one and it is not corrupted.</td>
</tr>
<tr>
<td>XSLAT.D</td>
<td>The log directory <code>&lt;directoryName&gt;</code> exists. The directory might belong to another database. Check that the location specified for the logDevice attribute is correct.</td>
</tr>
</tbody>
</table>
## Error Class XSLB: RawStore - Log.Generic statement exceptions

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLB1.S</td>
<td>Log operation <code>&lt;logOperation&gt;</code> 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.</td>
</tr>
<tr>
<td>XSLB2.S</td>
<td>Log operation <code>&lt;logOperation&gt;</code> logging excessive data, it filled up the internal log buffer.</td>
</tr>
<tr>
<td>XSLB4.S</td>
<td>Cannot find truncationLWM <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XSLB5.S</td>
<td>Illegal truncationLWM instant <code>&lt;value&gt;</code> for truncation point <code>&lt;value&gt;</code>. Legal range is from <code>&lt;value&gt;</code> to <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XSLB6.S</td>
<td>Trying to log a 0 or -ve length log Record.</td>
</tr>
<tr>
<td>XSLB8.S</td>
<td>Trying to reset a scan to <code>&lt;value&gt;</code>, beyond its limit of <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XSLB9.S</td>
<td>Cannot issue any more change, log factory has been stopped.</td>
</tr>
</tbody>
</table>
## Error Class XSRS: RawStore - protocol.Interface statement

### Error Class XSRS: RawStore - protocol.Interface statement

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSRS0.S</td>
<td>Cannot freeze the database after it is already frozen.</td>
</tr>
<tr>
<td>XSRS1.S</td>
<td>Cannot backup the database to <code>&lt;value&gt;</code>, which is not a directory.</td>
</tr>
<tr>
<td>XSRS4.S</td>
<td>Error renaming file (during backup) from <code>&lt;value&gt;</code> to <code>&lt;value&gt;</code>.</td>
</tr>
<tr>
<td>XSRS5.S</td>
<td>Error copying file (during backup) from <code>&lt;path&gt;</code> to <code>&lt;path&gt;</code>.</td>
</tr>
<tr>
<td>XSRS6.S</td>
<td>Cannot create backup directory <code>&lt;directoryName&gt;</code>.</td>
</tr>
<tr>
<td>XSRS7.S</td>
<td>Backup caught unexpected exception.</td>
</tr>
<tr>
<td>XSRS8.S</td>
<td>Log Device can only be set during database creation time, it cannot be changed on the fly.</td>
</tr>
<tr>
<td>XSRS9.S</td>
<td>Record <code>&lt;recordName&gt;</code> no longer exists</td>
</tr>
<tr>
<td>XSRSA.S</td>
<td>Cannot backup the database when unlogged operations are uncommitted. Please commit the transactions with backup blocking operations.</td>
</tr>
<tr>
<td>XSRSB.S</td>
<td>Backup cannot be performed in a transaction with uncommitted unlogged operations.</td>
</tr>
<tr>
<td>XSRSC.S</td>
<td>Cannot backup the database to <code>&lt;directoryLocation&gt;</code>, it is a database directory.</td>
</tr>
<tr>
<td>XSRS.D.S</td>
<td>Database backup is disabled. Contact your Splice Machine representative to enable.</td>
</tr>
<tr>
<td>XSRSE.S</td>
<td>Unable to enable the enterprise Manager. Enterprise services are disabled. Contact your Splice Machine representative to enable.</td>
</tr>
<tr>
<td>XSRSF.S</td>
<td>LDAP authentication is disabled. Contact your Splice Machine representative to enable.</td>
</tr>
<tr>
<td>XSRSG.S</td>
<td>SpliceMachine Enterprise services are disabled and so will not run on an encrypted host. Contact your Splice Machine representative to enable.</td>
</tr>
</tbody>
</table>
### Error Class XSTA2: XACT_TRANSACTION_ACTIVE

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSTA2.S</td>
<td>A transaction was already active, when attempt was made to make another transaction active.</td>
</tr>
</tbody>
</table>
Error Class XSTB: RawStore - Transactions.Basic system

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSTB0 .M</td>
<td>An exception was thrown during transaction abort.</td>
</tr>
<tr>
<td>XSTB2 .M</td>
<td>Cannot log transaction changes, maybe trying to write to a read only database.</td>
</tr>
<tr>
<td>XSTB3 .M</td>
<td>Cannot abort transaction because the log manager is null, probably due to an earlier error.</td>
</tr>
<tr>
<td>XSTB6 .M</td>
<td>Cannot substitute a transaction table with another while one is already in use.</td>
</tr>
</tbody>
</table>

## Error Class XXXXX: No SQLSTATE

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXXXX</td>
<td>Normal database session close.</td>
</tr>
</tbody>
</table>
Error Class X0 - Execution exceptions

<table>
<thead>
<tr>
<th>SQLSTATE</th>
<th>Message Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>X0A00.S</td>
<td>The select list mentions column '&lt;columnName&gt;' twice. This is not allowed in queries with GROUP BY or HAVING clauses. Try aliasing one of the conflicting columns to a unique name.</td>
</tr>
<tr>
<td>X0X02.S</td>
<td>Table '&lt;tableName&gt;' cannot be locked in '&lt;mode&gt;' mode.</td>
</tr>
<tr>
<td>X0X03.S</td>
<td>Invalid transaction state - held cursor requires same isolation level</td>
</tr>
<tr>
<td>X0X05.S</td>
<td>Table/View '&lt;tableName&gt;' does not exist.</td>
</tr>
<tr>
<td>X0X07.S</td>
<td>Cannot remove jar file '&lt;fileName&gt;' because it is on your derby.database.classpath '&lt;classpath&gt;'.</td>
</tr>
<tr>
<td>X0X0D.S</td>
<td>Invalid column array length '&lt;columnArrayLength&gt;'. To return generated keys, column array must be of length 1 and contain only the identity column.</td>
</tr>
<tr>
<td>X0X0E.S</td>
<td>Table '&lt;columnPosition&gt;' does not have an auto-generated column at column position '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>X0X0F.S</td>
<td>Table '&lt;columnName&gt;' does not have an auto-generated column named '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>X0X10.S</td>
<td>The USING clause returned more than one row; only single-row ResultSets are permissible.</td>
</tr>
<tr>
<td>X0X11.S</td>
<td>The USING clause did not return any results so no parameters can be set.</td>
</tr>
<tr>
<td>X0X13.S</td>
<td>Jar file '&lt;fileName&gt;' does not exist in schema '&lt;schemaName&gt;'.</td>
</tr>
<tr>
<td>X0X14.S</td>
<td>The file '&lt;fileName&gt;' does not exist.</td>
</tr>
<tr>
<td>X0X57.S</td>
<td>An attempt was made to put a Java value of type '&lt;type&gt;' 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.</td>
</tr>
<tr>
<td>X0X60.S</td>
<td>A cursor with name '&lt;cursorName&gt;' already exists.</td>
</tr>
<tr>
<td>X0X61.S</td>
<td>The values for column '&lt;columnName&gt;' in index '&lt;indexName&gt;' and table '&lt;schemaName&gt;.&lt;tableName&gt;' do not match for row location '&lt;location&gt;'. The value in the index is '&lt;value&gt;', while the value in the base table is '&lt;value&gt;'. The full index key, including the row location, is '&lt;indexKey&gt;'. The suggested corrective action is to recreate the index.</td>
</tr>
<tr>
<td>X0X62.S</td>
<td>Inconsistency found between table '&lt;tableName&gt;' and index '&lt;indexName&gt;'. Error when trying to retrieve row location '&lt;rowLocation&gt;' from the table. The full index key, including the row location, is '&lt;indexKey&gt;'. The suggested corrective action is to recreate the index.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>X0X63.S</td>
<td>Got IOException '&lt;value&gt;'.</td>
</tr>
<tr>
<td>X0X67.S</td>
<td>Columns of type '&lt;type&gt;' 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.</td>
</tr>
<tr>
<td>X0X81.S</td>
<td>'&lt;value&gt;' '&lt;value&gt;' does not exist.</td>
</tr>
<tr>
<td>X0X85.S</td>
<td>Index '&lt;indexName&gt;' was not created because '&lt;indexType&gt;' is not a valid index type.</td>
</tr>
<tr>
<td>X0X86.S</td>
<td>0 is an invalid parameter value for ResultSet.absolute(int row).</td>
</tr>
<tr>
<td>X0X87.S</td>
<td>ResultSet.relative(int row) cannot be called when the cursor is not positioned on a row.</td>
</tr>
<tr>
<td>X0X95.S</td>
<td>Operation '&lt;operationName&gt;' cannot be performed on object '&lt;objectName&gt;' because there is an open ResultSet dependent on that object.</td>
</tr>
<tr>
<td>X0X99.S</td>
<td>Index '&lt;indexName&gt;' does not exist.</td>
</tr>
<tr>
<td>X0Y16.S</td>
<td>'&lt;value&gt;' is not a view. If it is a table, then use DROP TABLE instead.</td>
</tr>
<tr>
<td>X0Y23.S</td>
<td>Operation '&lt;operationName&gt;' cannot be performed on object '&lt;objectName&gt;' because VIEW '&lt;viewName&gt;' is dependent on that object.</td>
</tr>
<tr>
<td>X0Y24.S</td>
<td>Operation '&lt;operationName&gt;' cannot be performed on object '&lt;objectName&gt;' because STATEMENT '&lt;statement&gt;' is dependent on that object.</td>
</tr>
<tr>
<td>X0Y25.S</td>
<td>Operation '&lt;operationName&gt;' cannot be performed on object '&lt;objectName&gt;' because '&lt;value&gt;' '&lt;value&gt;' is dependent on that object.</td>
</tr>
<tr>
<td>X0Y26.S</td>
<td>Index '&lt;indexName&gt;' is required to be in the same schema as table '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>X0Y28.S</td>
<td>Index '&lt;indexName&gt;' cannot be created on system table '&lt;tableName&gt;'. Users cannot create indexes on system tables.</td>
</tr>
<tr>
<td>X0Y29.S</td>
<td>Operation '&lt;operationName&gt;' cannot be performed on object '&lt;objectName&gt;' because TABLE '&lt;tableName&gt;' is dependent on that object.</td>
</tr>
<tr>
<td>X0Y30.S</td>
<td>Operation '&lt;operationName&gt;' cannot be performed on object '&lt;objectName&gt;' because ROUTINE '&lt;routineName&gt;' is dependent on that object.</td>
</tr>
<tr>
<td>X0Y32.S</td>
<td>'&lt;value&gt;' '&lt;value&gt;' already exists in '&lt;value&gt;' '&lt;value&gt;'.</td>
</tr>
<tr>
<td>X0Y38.S</td>
<td>Cannot create index '&lt;indexName&gt;' because table '&lt;tableName&gt;' does not exist.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>X0Y41.S</td>
<td>Constraint '&lt;constraintName&gt;' is invalid because the referenced table &lt;tableName&gt; has no primary key. Either add a primary key to &lt;tableName&gt; or explicitly specify the columns of a unique constraint that this foreign key references.</td>
</tr>
<tr>
<td>X0Y42.S</td>
<td>Constraint '&lt;constraintName&gt;' is invalid: the types of the foreign key columns do not match the types of the referenced columns.</td>
</tr>
<tr>
<td>X0Y43.S</td>
<td>Constraint '&lt;constraintName&gt;' is invalid: the number of columns in &lt;value&gt; (&lt;value&gt;) does not match the number of columns in the referenced key (&lt;value&gt;).</td>
</tr>
<tr>
<td>X0Y44.S</td>
<td>Constraint '&lt;constraintName&gt;' is invalid: there is no unique or primary key constraint on table '&lt;tableName&gt;' that matches the number and types of the columns in the foreign key.</td>
</tr>
<tr>
<td>X0Y45.S</td>
<td>Foreign key constraint '&lt;constraintName&gt;' cannot be added to or enabled on table &lt;tableName&gt; because one or more foreign keys do not have matching referenced keys.</td>
</tr>
<tr>
<td>X0Y46.S</td>
<td>Constraint '&lt;constraintName&gt;' is invalid: referenced table &lt;tableName&gt; does not exist.</td>
</tr>
<tr>
<td>X0Y54.S</td>
<td>Schema '&lt;schemaName&gt;' cannot be dropped because it is not empty.</td>
</tr>
<tr>
<td>X0Y55.S</td>
<td>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 '&lt;indexName&gt;' on table '&lt;schemaName&gt;.&lt;tableName&gt;' has &lt;number&gt; rows, but the base table has &lt;number&gt; rows. The suggested corrective action is to recreate the index.</td>
</tr>
<tr>
<td>X0Y56.S</td>
<td>'&lt;value&gt;' is not allowed on the System table '&lt;tableName&gt;'.</td>
</tr>
<tr>
<td>X0Y57.S</td>
<td>A non-nullable column cannot be added to table '&lt;tableName&gt;' because the table contains at least one row. Non-nullable columns can only be added to empty tables.</td>
</tr>
<tr>
<td>X0Y58.S</td>
<td>Attempt to add a primary key constraint to table '&lt;tableName&gt;' failed because the table already has a constraint of that type. A table can only have a single primary key constraint.</td>
</tr>
<tr>
<td>X0Y59.S</td>
<td>Attempt to add or enable constraint(s) on table '&lt;tableName&gt;' failed because the table contains &lt;rowName&gt; row(s) that violate the following check constraint(s): &lt;constraintName&gt;.</td>
</tr>
<tr>
<td>X0Y63.S</td>
<td>The command on table '&lt;tableName&gt;' 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.</td>
</tr>
<tr>
<td>X0Y63.S.1</td>
<td>The command on table '&lt;tableName&gt;' failed because null data was found in the primary key/index column(s). All columns in a primary key must not be null.</td>
</tr>
<tr>
<td>X0Y66.S</td>
<td>Cannot issue commit in a nested connection when there is a pending operation in the parent connection.</td>
</tr>
<tr>
<td>X0Y67.S</td>
<td>Cannot issue rollback in a nested connection when there is a pending operation in the parent connection.</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>Message Text</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>X0Y68.S</td>
<td>&lt;value&gt; 'value' already exists.</td>
</tr>
<tr>
<td>X0Y69.S</td>
<td>&lt;value&gt; is not supported in trigger &lt;triggerName&gt;.</td>
</tr>
<tr>
<td>X0Y70.S</td>
<td>INSERT, UPDATE and DELETE are not permitted on table &lt;tableName&gt; because trigger &lt;triggerName&gt; is active.</td>
</tr>
<tr>
<td>X0Y71.S</td>
<td>Transaction manipulation such as SET ISOLATION is not permitted because trigger &lt;triggerName&gt; is active.</td>
</tr>
<tr>
<td>X0Y72.S</td>
<td>Bulk insert replace is not permitted on '&lt;value&gt;' because it has an enabled trigger ('value').</td>
</tr>
<tr>
<td>X0Y77.S</td>
<td>Cannot issue set transaction isolation statement on a global transaction that is in progress because it would have implicitly committed the global transaction.</td>
</tr>
<tr>
<td>X0Y78.S</td>
<td>Statement.executeQuery() cannot be called with a statement that returns a row count.</td>
</tr>
<tr>
<td>X0Y78.S.1</td>
<td>&lt;value&gt;.executeQuery() cannot be called because multiple result sets were returned. Use &lt;value&gt;.execute() to obtain multiple results.</td>
</tr>
<tr>
<td>X0Y78.S.2</td>
<td>&lt;value&gt;.executeQuery() was called but no result set was returned. Use &lt;value&gt;.executeUpdate() for non-queries.</td>
</tr>
<tr>
<td>X0Y79.S</td>
<td>Statement.executeUpdate() cannot be called with a statement that returns a ResultSet.</td>
</tr>
<tr>
<td>X0Y80.S</td>
<td>ALTER table '&lt;tableName&gt;' failed. Null data found in column '&lt;columnName&gt;'.</td>
</tr>
<tr>
<td>X0Y83.S</td>
<td>WARNING: While deleting a row from a table the index row for base table row &lt;rowName&gt; was not found in index with conglomerate id &lt;id&gt;. This problem has automatically been corrected as part of the delete operation.</td>
</tr>
<tr>
<td>X0Y84.T</td>
<td>Too much contention on sequence &lt;sequenceName&gt;. This is probably caused by an uncommitted scan of the SYS.SYSEQUENCES 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.</td>
</tr>
<tr>
<td>X0Y85.S</td>
<td>The Splice property '&lt;propertyName&gt;' identifies a class which cannot be instantiated: '&lt;className&gt;'. See the next exception for details.</td>
</tr>
<tr>
<td>X0Y86.S</td>
<td>Splice could not obtain the locks needed to release the unused, preallocated values for the sequence '&lt;schemaName&gt;','&lt;sequenceName&gt;'. As a result, unexpected gaps may appear in this sequence.</td>
</tr>
<tr>
<td>X0Y87.S</td>
<td>There is already an aggregate or function with one argument whose name is '&lt;schemaName&gt;','&lt;aggregateOrFunctionName&gt;'.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>If exactly one Java method matches, then Splice Machine invokes it.</td>
</tr>
<tr>
<td>Ambiguity</td>
<td>If exactly one Java method matches, then Splice Machine invokes it.</td>
</tr>
<tr>
<td>Failure</td>
<td>Splice Machine also raises an error if no method matches.</td>
</tr>
</tbody>
</table>

In mapping SQL data types to Java data types, Splice Machine considers the following kinds of matches:

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Primitive Match</strong></td>
<td>Splice Machine looks for a primitive Java type corresponding to the SQL type. For instance, SQL INTEGER matches Java int</td>
</tr>
<tr>
<td><strong>Wrapper Match</strong></td>
<td>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 INTEGER matches <code>java.lang.Integer</code>. For a user-defined type (UDT), Splice Machine looks for the UDT's external name class.</td>
</tr>
<tr>
<td><strong>Array Match</strong></td>
<td>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 INTEGER matches <code>int[]</code> and <code>Integer[]</code>.</td>
</tr>
<tr>
<td><strong>ResultSet Match</strong></td>
<td>If a procedure is declared to return <code>n</code> RESULT SETS, Splice Machine looks for a method whose last <code>n</code> arguments are of type <code>java.sql.ResultSet[]</code>.</td>
</tr>
</tbody>
</table>

Splice Machine resolves function and procedure invocations as follows:
Call type | Resolution
---|---
**Function** | Splice Machine looks for a method whose argument and return types are *primitive matches or wrapper matches* for the function's SQL arguments and return value.

**Procedure** | Splice Machine looks for a method which returns void and whose argument types match as follows:

- **IN** - Method arguments are *primitive matches or wrapper matches* for the procedure's IN arguments.
- **OUT** and **INOUT** - Method arguments are *array matches* for the procedure's OUT and INOUT arguments.

In addition, if the procedure returns $n$ RESULT SETS, then the last $n$ arguments of the Java method must be of type `java.sql.ResultSet[]`

---

**Example of argument matching**

The following function:

```sql
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:

```java
public static double toDegrees( double arg ) {...}
public static Double toDegrees( double arg ) {...}
public static double toDegrees( Double arg ) {...}
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.
<table>
<thead>
<tr>
<th>SQL Type</th>
<th>Primitive Match</th>
<th>Wrapper Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>boolean</td>
<td>java.lang.Boolean</td>
</tr>
<tr>
<td>TINYINT</td>
<td>byte</td>
<td>java.lang.Byte</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>short</td>
<td>java.lang.Integer</td>
</tr>
<tr>
<td>INTEGER</td>
<td>int</td>
<td>java.lang.Integer</td>
</tr>
<tr>
<td>BIGINT</td>
<td>long</td>
<td>java.lang.Long</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>None</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>None</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>REAL</td>
<td>float</td>
<td>java.lang.Float</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>double</td>
<td>java.lang.Double</td>
</tr>
<tr>
<td>FLOAT</td>
<td>double</td>
<td>java.lang.Double</td>
</tr>
<tr>
<td>CHAR</td>
<td>None</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>None</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>None</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>CLOB</td>
<td>None</td>
<td>java.sql.Clob</td>
</tr>
<tr>
<td>BLOB</td>
<td>None</td>
<td>java.sql.Blob</td>
</tr>
<tr>
<td>DATE</td>
<td>None</td>
<td>java.sql.Date</td>
</tr>
<tr>
<td>TIME</td>
<td>None</td>
<td>java.sql.Time</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>None</td>
<td>java.sql.Timestamp</td>
</tr>
<tr>
<td>User-defined type</td>
<td>None</td>
<td>Underlying Java class</td>
</tr>
</tbody>
</table>

**See Also**

» About Data Types
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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum columns in a table</td>
<td>131072</td>
</tr>
<tr>
<td>Maximum columns in a view</td>
<td>5000</td>
</tr>
<tr>
<td>Maximum number of parameters in a stored procedure</td>
<td>90</td>
</tr>
<tr>
<td>Maximum indexes on a table</td>
<td>32767 or storage capacity</td>
</tr>
<tr>
<td>Maximum tables referenced in an SQL statement or a view</td>
<td>Storage capacity</td>
</tr>
<tr>
<td>Maximum elements in a select list</td>
<td>10000</td>
</tr>
<tr>
<td>Maximum predicates in a WHERE or HAVING clause</td>
<td>Storage capacity</td>
</tr>
<tr>
<td>Maximum number of columns in a GROUP BY clause</td>
<td>32677</td>
</tr>
<tr>
<td>Maximum number of columns in an ORDER BY clause</td>
<td>1012</td>
</tr>
<tr>
<td>Maximum number of prepared statements</td>
<td>Storage capacity</td>
</tr>
<tr>
<td>Maximum declared cursors in a program</td>
<td>Storage capacity</td>
</tr>
<tr>
<td>Maximum number of cursors opened at one time</td>
<td>Storage capacity</td>
</tr>
<tr>
<td>Maximum number of constraints on a table</td>
<td>Storage capacity</td>
</tr>
</tbody>
</table>
### Date, Time, and TimeStamp Limitations
The following table lists limitations on date, time, and timestamp values in Splice Machine.

<table>
<thead>
<tr>
<th>Value</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smallest DATE value</td>
<td>0001-01-01</td>
</tr>
<tr>
<td>Largest DATE value</td>
<td>9999-12-31</td>
</tr>
<tr>
<td>Smallest TIME value</td>
<td>00:00:00</td>
</tr>
<tr>
<td>Largest TIME value</td>
<td>24:00:00</td>
</tr>
<tr>
<td>Smallest TIMESTAMP value</td>
<td>0000-00-00-00.00.00.00.000000</td>
</tr>
<tr>
<td>Largest TIMESTAMP value</td>
<td>9999-12-31-23.59.59.999999</td>
</tr>
</tbody>
</table>

### Identifier Length Limitations
The following table lists limitations on identifier lengths in Splice Machine.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Maximum Number of Characters Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint name</td>
<td>128</td>
</tr>
<tr>
<td>Correlation name</td>
<td>128</td>
</tr>
<tr>
<td>Cursor name</td>
<td>128</td>
</tr>
</tbody>
</table>
### Numeric Limitations

The following lists limitations on the numeric values in Splice Machine.

<table>
<thead>
<tr>
<th>Value</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smallest INTEGER</td>
<td>(-2,147,483,648)</td>
</tr>
<tr>
<td>Largest INTEGER</td>
<td>(2,147,483,647)</td>
</tr>
<tr>
<td>Smallest BIGINT</td>
<td>(-9,223,372,036,854,775,808)</td>
</tr>
<tr>
<td>Largest BIGINT</td>
<td>(9,223,372,036,854,775,807)</td>
</tr>
<tr>
<td>Smallest SMALLINT</td>
<td>(-32,768)</td>
</tr>
<tr>
<td>Largest SMALLINT</td>
<td>(32,767)</td>
</tr>
<tr>
<td>Value</td>
<td>Limit</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Smallest TINYINT</td>
<td>-128</td>
</tr>
<tr>
<td>Largest TINYINT</td>
<td>127</td>
</tr>
<tr>
<td>Largest decimal precision</td>
<td>31</td>
</tr>
<tr>
<td>Smallest DOUBLE</td>
<td>-1.79769E+308</td>
</tr>
<tr>
<td>Largest DOUBLE</td>
<td>1.79769E+308</td>
</tr>
<tr>
<td>Smallest positive DOUBLE</td>
<td>2.225E-307</td>
</tr>
<tr>
<td>Largest negative DOUBLE</td>
<td>-2.225E-307</td>
</tr>
<tr>
<td>Smallest REAL</td>
<td>-3.402E+38</td>
</tr>
<tr>
<td>Largest REAL</td>
<td>3.402E+38</td>
</tr>
<tr>
<td>Smallest positive REAL</td>
<td>1.175E-37</td>
</tr>
<tr>
<td>Largest negative REAL</td>
<td>-1.175E-37</td>
</tr>
</tbody>
</table>

**String Limitations**
The following table lists limitations on string values in Splice Machine.

<table>
<thead>
<tr>
<th>Value</th>
<th>Maximum Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of CHAR</td>
<td>254 characters</td>
</tr>
<tr>
<td>Length of VARCHAR</td>
<td>32,672 characters</td>
</tr>
<tr>
<td>Length of LONG VARCHAR</td>
<td>32,670 characters</td>
</tr>
<tr>
<td>Length of CLOB*</td>
<td>2,147,483,647 characters</td>
</tr>
<tr>
<td>Length of BLOB*</td>
<td>2,147,483,647 characters</td>
</tr>
<tr>
<td>Length of character constant</td>
<td>32,672</td>
</tr>
<tr>
<td>Length of concatenated character string</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>Value</td>
<td>Maximum Limit</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Length of concatenated binary string</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>Number of hex constant digits</td>
<td>16,336</td>
</tr>
<tr>
<td>Length of DOUBLE value constant</td>
<td>30 characters</td>
</tr>
</tbody>
</table>

* 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.
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.

<table>
<thead>
<tr>
<th>Reserved Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
</tr>
<tr>
<td>ALL</td>
</tr>
<tr>
<td>ALLOCATE</td>
</tr>
<tr>
<td>ALTER</td>
</tr>
<tr>
<td>AND</td>
</tr>
<tr>
<td>ANY</td>
</tr>
<tr>
<td>ARE</td>
</tr>
<tr>
<td>AS</td>
</tr>
<tr>
<td>ASC</td>
</tr>
<tr>
<td>ASSERTION</td>
</tr>
<tr>
<td>AT</td>
</tr>
<tr>
<td>AUTHORIZATION</td>
</tr>
<tr>
<td>AVG</td>
</tr>
<tr>
<td>BEGIN</td>
</tr>
<tr>
<td>BETWEEN</td>
</tr>
<tr>
<td>BIGINT</td>
</tr>
<tr>
<td>BIT</td>
</tr>
<tr>
<td>BOOLEAN</td>
</tr>
<tr>
<td>BOTH</td>
</tr>
<tr>
<td>BY</td>
</tr>
<tr>
<td>Reserved Word</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>CALL</td>
</tr>
<tr>
<td>CASCADE</td>
</tr>
<tr>
<td>CASCADED</td>
</tr>
<tr>
<td>CASE</td>
</tr>
<tr>
<td>CAST</td>
</tr>
<tr>
<td>CHAR</td>
</tr>
<tr>
<td>CHARACTER</td>
</tr>
<tr>
<td>CHECK</td>
</tr>
<tr>
<td>CLOSE</td>
</tr>
<tr>
<td>COALESCE</td>
</tr>
<tr>
<td>COLLATE</td>
</tr>
<tr>
<td>COLLATION</td>
</tr>
<tr>
<td>COLUMN</td>
</tr>
<tr>
<td>COMMIT</td>
</tr>
<tr>
<td>CONNECT</td>
</tr>
<tr>
<td>CONNECTION</td>
</tr>
<tr>
<td>CONSTRAINT</td>
</tr>
<tr>
<td>CONSTRAINTS</td>
</tr>
<tr>
<td>CONTINUE</td>
</tr>
<tr>
<td>CONVERT</td>
</tr>
<tr>
<td>CORRESPONDING</td>
</tr>
<tr>
<td>CREATE</td>
</tr>
<tr>
<td>CROSS</td>
</tr>
<tr>
<td>CURRENT</td>
</tr>
<tr>
<td>Reserved Word</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td>CURRENT_DATE</td>
</tr>
<tr>
<td>CURRENT_ROLE</td>
</tr>
<tr>
<td>CURRENT_TIME</td>
</tr>
<tr>
<td>CURRENT_TIMESTAMP</td>
</tr>
<tr>
<td>CURRENT_USER</td>
</tr>
<tr>
<td>CURSOR</td>
</tr>
<tr>
<td>DEALLOCATE</td>
</tr>
<tr>
<td>DEC</td>
</tr>
<tr>
<td>DECIMAL</td>
</tr>
<tr>
<td>DECLARE</td>
</tr>
<tr>
<td>DEFAULT</td>
</tr>
<tr>
<td>DEFERRABLE</td>
</tr>
<tr>
<td>DEFERRED</td>
</tr>
<tr>
<td>DEFERRED</td>
</tr>
<tr>
<td>DELETE</td>
</tr>
<tr>
<td>DESC</td>
</tr>
<tr>
<td>DESCRIBE</td>
</tr>
<tr>
<td>DIAGNOSTICS</td>
</tr>
<tr>
<td>DISCONNECT</td>
</tr>
<tr>
<td>DISTINCT</td>
</tr>
<tr>
<td>DOUBLE</td>
</tr>
<tr>
<td>DROP</td>
</tr>
<tr>
<td>ELSE</td>
</tr>
<tr>
<td>END</td>
</tr>
<tr>
<td>END-EXEC</td>
</tr>
<tr>
<td>Reserved Word</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>ESCAPE</td>
</tr>
<tr>
<td>EXCEPT</td>
</tr>
<tr>
<td>EXCEPTION</td>
</tr>
<tr>
<td>EXEC</td>
</tr>
<tr>
<td>EXECUTE</td>
</tr>
<tr>
<td>EXISTS</td>
</tr>
<tr>
<td>EXPLAIN</td>
</tr>
<tr>
<td>EXTERNAL</td>
</tr>
<tr>
<td>FALSE</td>
</tr>
<tr>
<td>FETCH</td>
</tr>
<tr>
<td>FIRST</td>
</tr>
<tr>
<td>FLOAT</td>
</tr>
<tr>
<td>FOR</td>
</tr>
<tr>
<td>FOREIGN</td>
</tr>
<tr>
<td>FOUND</td>
</tr>
<tr>
<td>FROM</td>
</tr>
<tr>
<td>FULL</td>
</tr>
<tr>
<td>FUNCTION</td>
</tr>
<tr>
<td>GET</td>
</tr>
<tr>
<td>GETCURRENTCONNECTION</td>
</tr>
<tr>
<td>GLOBAL</td>
</tr>
<tr>
<td>GO</td>
</tr>
<tr>
<td>GOTO</td>
</tr>
<tr>
<td>GRANT</td>
</tr>
<tr>
<td>Reserved Word</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>GROUP</td>
</tr>
<tr>
<td>HAVING</td>
</tr>
<tr>
<td>HOUR</td>
</tr>
<tr>
<td>IDENTITY</td>
</tr>
<tr>
<td>IMMEDIATE</td>
</tr>
<tr>
<td>IN</td>
</tr>
<tr>
<td>INDICATOR</td>
</tr>
<tr>
<td>INITIALLY</td>
</tr>
<tr>
<td>INNER</td>
</tr>
<tr>
<td>INOUT</td>
</tr>
<tr>
<td>INPUT</td>
</tr>
<tr>
<td>INSENSITIVE</td>
</tr>
<tr>
<td>INSERT</td>
</tr>
<tr>
<td>INT</td>
</tr>
<tr>
<td>INTEGER</td>
</tr>
<tr>
<td>INTERSECT</td>
</tr>
<tr>
<td>INTO</td>
</tr>
<tr>
<td>IS</td>
</tr>
<tr>
<td>ISOLATION</td>
</tr>
<tr>
<td>JOIN</td>
</tr>
<tr>
<td>KEY</td>
</tr>
<tr>
<td>LAST</td>
</tr>
<tr>
<td>LEFT</td>
</tr>
<tr>
<td>LIKE</td>
</tr>
<tr>
<td>Reserved Word</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>LOWER</td>
</tr>
<tr>
<td>LTRIM</td>
</tr>
<tr>
<td>MATCH</td>
</tr>
<tr>
<td>MAX</td>
</tr>
<tr>
<td>MIN</td>
</tr>
<tr>
<td>MINUTE</td>
</tr>
<tr>
<td>NATIONAL</td>
</tr>
<tr>
<td>NATURAL</td>
</tr>
<tr>
<td>NCHAR</td>
</tr>
<tr>
<td>NVARCHAR</td>
</tr>
<tr>
<td>NEXT</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>NONE</td>
</tr>
<tr>
<td>NOT</td>
</tr>
<tr>
<td>NULL</td>
</tr>
<tr>
<td>NULLIF</td>
</tr>
<tr>
<td>NUMERIC</td>
</tr>
<tr>
<td>OF</td>
</tr>
<tr>
<td>ON</td>
</tr>
<tr>
<td>ONLY</td>
</tr>
<tr>
<td>OPEN</td>
</tr>
<tr>
<td>OPTION</td>
</tr>
<tr>
<td>OR</td>
</tr>
<tr>
<td>ORDER</td>
</tr>
<tr>
<td>Reserved Word</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>OUTER</td>
</tr>
<tr>
<td>OUTPUT</td>
</tr>
<tr>
<td>OVER</td>
</tr>
<tr>
<td>OVERLAPS</td>
</tr>
<tr>
<td>PAD</td>
</tr>
<tr>
<td>PARTIAL</td>
</tr>
<tr>
<td>PREPARE</td>
</tr>
<tr>
<td>PRESERVE</td>
</tr>
<tr>
<td>PRIMARY</td>
</tr>
<tr>
<td>PRIOR</td>
</tr>
<tr>
<td>PRIVILEGES</td>
</tr>
<tr>
<td>PROCEDURE</td>
</tr>
<tr>
<td>PUBLIC</td>
</tr>
<tr>
<td>READ</td>
</tr>
<tr>
<td>REAL</td>
</tr>
<tr>
<td>REFERENCES</td>
</tr>
<tr>
<td>REGEXP_LIKE</td>
</tr>
<tr>
<td>RELATIVE</td>
</tr>
<tr>
<td>RESTRICT</td>
</tr>
<tr>
<td>REVOKE</td>
</tr>
<tr>
<td>RIGHT</td>
</tr>
<tr>
<td>ROLLBACK</td>
</tr>
<tr>
<td>ROWS</td>
</tr>
<tr>
<td>ROW_NUMBER</td>
</tr>
<tr>
<td>Reserved Word</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>RTRIM</td>
</tr>
<tr>
<td>SCHEMA</td>
</tr>
<tr>
<td>SCROLL</td>
</tr>
<tr>
<td>SECOND</td>
</tr>
<tr>
<td>SELECT</td>
</tr>
<tr>
<td>SESSION_USER</td>
</tr>
<tr>
<td>SET</td>
</tr>
<tr>
<td>SMALLINT</td>
</tr>
<tr>
<td>SOME</td>
</tr>
<tr>
<td>SPACE</td>
</tr>
<tr>
<td>SQL</td>
</tr>
<tr>
<td>SPCODE</td>
</tr>
<tr>
<td>SQLERROR</td>
</tr>
<tr>
<td>SQLSTATE</td>
</tr>
<tr>
<td>SUBSTR</td>
</tr>
<tr>
<td>SUBSTRING</td>
</tr>
<tr>
<td>SUM</td>
</tr>
<tr>
<td>TABLE</td>
</tr>
<tr>
<td>TEMPORARY</td>
</tr>
<tr>
<td>TEXT</td>
</tr>
<tr>
<td>TIMEZONE_HOUR</td>
</tr>
<tr>
<td>TIMEZONE_MINUTE</td>
</tr>
<tr>
<td>TINYINT</td>
</tr>
<tr>
<td>TO</td>
</tr>
<tr>
<td>Reserved Word</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>TRANSACTION</td>
</tr>
<tr>
<td>TRANSLATE</td>
</tr>
<tr>
<td>TRANSLATION</td>
</tr>
<tr>
<td>TRIM</td>
</tr>
<tr>
<td>TRUE</td>
</tr>
<tr>
<td>UNION</td>
</tr>
<tr>
<td>UNIQUE</td>
</tr>
<tr>
<td>UNKNOWN</td>
</tr>
<tr>
<td>UPDATE</td>
</tr>
<tr>
<td>UPPER</td>
</tr>
<tr>
<td>USER</td>
</tr>
<tr>
<td>USING</td>
</tr>
<tr>
<td>VALUES</td>
</tr>
<tr>
<td>VARCHAR</td>
</tr>
<tr>
<td>VARYING</td>
</tr>
<tr>
<td>VIEW</td>
</tr>
<tr>
<td>WHENEVER</td>
</tr>
<tr>
<td>WHERE</td>
</tr>
<tr>
<td>WITH</td>
</tr>
<tr>
<td>WORK</td>
</tr>
<tr>
<td>WRITE</td>
</tr>
<tr>
<td>XML</td>
</tr>
<tr>
<td>XML EXISTS</td>
</tr>
<tr>
<td>XML PARSE</td>
</tr>
<tr>
<td>Reserved Word</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>XMLQUERY</td>
</tr>
<tr>
<td>XMLSERIALIZE</td>
</tr>
<tr>
<td>YEAR</td>
</tr>
</tbody>
</table>
**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:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aggregation functions</strong></td>
<td>AVG, COUNT, MAX, MIN, STDDEV_POP, STDDEV_SAMP, SUM</td>
</tr>
<tr>
<td><strong>Conditional functions</strong></td>
<td>CASE, searched CASE</td>
</tr>
<tr>
<td><strong>Data Types</strong></td>
<td>INTEGER, REAL, CHARACTER, DATE, BOOLEAN, BIGINT</td>
</tr>
<tr>
<td><strong>DDL</strong></td>
<td>CREATE TABLE, CREATE SCHEMA, CREATE INDEX, ALTER TABLE, DELETE, UPDATE</td>
</tr>
<tr>
<td><strong>DML</strong></td>
<td>INSERT, DELETE, UPDATE, SELECT</td>
</tr>
<tr>
<td><strong>Isolation Levels</strong></td>
<td>Snapshot isolation</td>
</tr>
<tr>
<td><strong>Joins</strong></td>
<td>INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN</td>
</tr>
<tr>
<td><strong>Predicates</strong></td>
<td>IN, BETWEEN, LIKE, EXISTS</td>
</tr>
<tr>
<td><strong>Privileges</strong></td>
<td>Privileges for SELECT, DELETE, INSERT, EXECUTE</td>
</tr>
<tr>
<td><strong>Query Specification</strong></td>
<td>SELECT DISTINCT, GROUP BY, HAVING</td>
</tr>
<tr>
<td><strong>SET functions</strong></td>
<td>UNION, ABS, MOD, ALL, CHECK</td>
</tr>
<tr>
<td><strong>String functions</strong></td>
<td>CHAR, Concatenation (</td>
</tr>
<tr>
<td><strong>Sub-queries</strong></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Transactions</strong></td>
<td>COMMIT, ROLLBACK</td>
</tr>
</tbody>
</table>
### 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