Command Line Interface Reference

Last generated: March 01, 2018

Table of Contents

Splice Machine Commands Reference

This section contains a reference topic page for each Splice Machine command. Another topic, Using the splice> Command Line Interface, presents general syntax and usage help for the `splice>` prompt.

- Commands you can use with all connections to Splice Machine databases, which means that you can use them with the `sqlshell` interface in our *On-Premise Database* and *Database-as-Service* products, and also with programs that connect to a Splice Machine database using JDBC or ODBC, including the Zeppelin notebook interface in our Database Service.
- Commands you can only use with the `splice>` (`sqlshell.sh`) command line interface in our *On-Premise Database* and *Database-as-Service* products. These are mostly commands that display information about the database in a terminal interface, and are not available through JDBC or ODBC connections.

Commands You Can Use with All Connections to a Splice Machine Database

The following table sunmmarizes the commands that you can use them with the `sqlshell` interface in our *On-Premise Database* and *Database-as-Service* products, and also with programs that connect to a Splice Machine database using JDBC or ODBC, including the Zeppelin notebook interface in our Database Service.

Command	Description	Usage
Analyze	Collects statistics for a table or schema.	`splice> analyze table myTable;` `splice> analyze schema myschema;`
Autocommit	Turns the connection's auto-commit mode on or off.	`splice> autocommit off;`
Commit	Commits the currently active transaction and initiates a new transaction.	`splice> commit;`
Execute	Executes an SQL prepared statement or SQL command string.	`splice> execute 'insert into myTable(id, val) values(?,?)' ;`
Explain	Displays the execution plan for an SQL statement.	`splice> explain select count(*) from si;`
Export	Exports query results to CSV files.	`splice> EXPORT('/my/export/dir', null, null, null, null, null) SELECT a,b,sqrt(c) FROM join t2 on t1.a=t2.a;`
Prepare	Creates a prepared statement for use by other commands.	`splice> prepare seeMenu as 'SELECT * FROM menu';`

Command	Description	Usage
Release Savepoint	Releases a savepoint.	`splice> release savepoint gSavePt1;`
Remove	Removes a previously prepared statement.	`splice> remove seeMenu;`
Rollback	Rolls back the currently active transaction and initiates a new transaction.	`splice> rollback;`
Rollback to Savepoint	Rolls the current transaction back to the specified savepoint.	`splice> rollback to savepoint gSavePt1;`
Savepoint	Creates a savepoint within the current transaction.	`splice> savepoint gSavePt1;`

Commands You Can Only Use with Our Command Line Interface (`sqlshell.sh`)

The following table sunmmarizes the commands that you can only use with the `splice>` command line interface (`sqlshell.sh`) in our *On-Premise Database* and *Database-as-Service* products.

Command	Description	Usage
Connect	Connect to a database via its URL.	`splice> connect 'jdbc:splice://xyz:1527/splicedb';`
Describe	Displays a description of a table or view.	`splice> describe myTable;`
Disconnect	Disconnects from a database.	`splice> disconnect SPLICE;`
Elapsedtime	Enables or disables display of elapsed time for command execution.	`splice> elapsedtime on;`
Exit	Causes the command line interface to exit.	`splice> exit;`
Help	Displays a list of the available commands.	`splice> help;`

Command	Description	Usage
MaximumDisplayWidth	Sets the maximum displayed width for each column of results displayed by the command line interpreter.	`splice> maximumdisplaywidth 30;`
Run	Runs commands from a file.	`splice> run myCmdFile;`
Set Connection	Allows you to specify which connection is the current connection	`splice> set connection sample1;`
Show Connections	Displays information about active connections and database objects.	`splice> show connections;`
Show Functions	Displays information about functions defined in the database or in a schema.	`splice> show functions in splice;`
Show Indexes	Displays information about the indexes defined on a table, a database, or a schema.	`splice> show indexes from mytable;`
Show Primary Keys	Displays information about the primary keys in a table.	`splice> show primarykeys from mySchema.myTable;`
Show Procedures	Displays information about active connections and database objects.	`splice> show procedures in syscs_util;`
Show Roles	Displays information about all of the roles defined in the database.	`splice> show roles;`
Show Schemas	Displays information about the schemas in the current connection.	`splice> show schemas;`
Show Synonyms	Displays information about the synonyms that have been created in a database or schema.	`splice> show synonyms;`
Show Tables	Displays information about all of the tables in a database or schema.	`splice> show tables in SPLICE;`
Show Views	Displays information about all of the active views in a schema.	`splice> show views in SPLICE;`

Using the splice> Command Line Interface

This topic presents information that will help you in using the Splice Machine `splice>` command line interpreter, in the following sections:

- The splice> Command Line Interpreter section shows you how to invoke the splice> command line.
- The Command Line Output section describes how you can adjust the appearance of output from the interepreter.
- The Command Line Syntax section summarizes the syntax of commands, including capitalization and case-sensitivity rules, as well as various special characters you can use in your commands. It also shows you how to include comments in your command lines and how to run a file of SQL commands.
- The Example Command Lines section shows several examples of command lines.
- The Scripting Splice Commands tutorial describes how to create a script of `splice>` commands to run a series of operations like loading a number of files into your database

The remainder of this section contains a reference page for each of the command line commands.

splice> Command Line Interpreter

To run the Splice Machine command line interpreter, run the `sqlshell.sh` script in your terminal window.

```
% ./sqlshell.sh
splice>
```

When the interpreter prompts you with `splice>`, you can enter commands, ending each with a semicolon. For a complete description of `splice>` syntax, see the next section in this topic, Command Line Syntax,

Note that you can optionally specify a file of sql commands that the interpreter will run using the `-f` parameter; after running those commands, the interpreter exits. For example:

```
./sqlshell.sh -f /home/mydir/sql/test.sql
```

You can optionally include parameter values when running `sqlshell.sh` script, to change default values. Here's the syntax:

```
sqlshell.sh [-h host] [-p port ] [-u username] [-s password] [-f commandsFile
```

-host
: The hostname or IP address of your Splice Machine HBase RegionServer.

  The default value is `localhost`.

-port
: The port on which Splice Machine is listening for your connection.

  The default value is `1527`.

-username
: The user name for your Splice Machine database.

  The default value is `splice`.

-password
: The password for your Splice Machine database.

  The default value is `admin`.

-f [fileName]
: The name of a file with SQL commands in it: `sqlshell` starts up the `splice>` command line interpreter, runs the commands in the file, and then exits. For example:

```
$ ./sqlshell.sh -f /home/mydir/sql/test.sql

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

Running Splice Machine SQL shell
For help: "splice> help;"
SPLICE* -         jdbc:splice://10.1.1.111:1527/splicedb
* = current connection
splice> elapsedtime on;
splice> select count(*) from CUST_EMAIL;
1
--------------------
0

1 row selected
ELAPSED TIME = 6399 milliseconds
splice>
$
```

The `test.sql` file used in the above example contains the following commands:

```
elapsedtime on;
select count(*) from CUST_EMAIL;
```

Command Line Output

Output from `splice>` commands is displayed in your terminal window. The `maximumdisplaywidth` setting affects how the output is displayed; specifically, it determines if the content of each column is truncated to fit within the width that you specify.

When you set `maximumdisplaywidth` to `0`, all output is displayed, without truncation.

Command Line Syntax

This section briefly summarizes the syntax of command lines you can enter in Zeppelin notebooks and in response to the `splice>` prompt, including these subsections:

Finish Commands with a Semicolon

The command line interface allows you to enter multi-line commands, and waits for a non-escaped semicolon (`;`) character to signal the end of the command.

A command is not executed until you enter the semicolon character and press the `Return` or `Enter` key.

Capitalization and Case Sensitivity Rules

Certain identifiers and keywords are case sensitive:

Identifier	Case Sensitive?	Notes and Example
SQL keywords	Not case sensitive	These are all equivalent: SELECT, Select, select, SeLeCt.
ANSI SQL identifiers	Not case sensitive	These are not case sensitive unless they are delimited.
Java-style identifiers	Always case sensitive	These are NOT equivalent: my_name, My_Name.

Special Characters You Can Use

The following table describes the special characters you can use in commands:

Purpose	Character(s) to use	Notes and example
To delimit special identifiers	Double quotation marks (")	Special identifiers are also known as *delimited identifiers*.
To delimit character strings	Single quotation marks (')	
To escape a single quote or apostrophe within a character string	Single quotation mark ((')	Since single quotation marks are used to delimit strings, you must escape any single quotation marks you want included in the string itself. 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. Example: 'This string includes "my quoted string" within it.'
To escape a double quote	*Not needed*	You can simply include a double quotation mark in your command lines.
To specify a wild card within a Select expression	The asterisk (*) character	This is the SQL metasymbol for selecting all matching entries. Example: SELECT * FROM MyTable;
To specify a wild card sequence in a string with the `LIKE` operator	The percentage (%) character	Example: SELECT * FROM MyTable WHERE Name LIKE 'Ga%';
To specify a single wild card character in a string with the `LIKE` operator	The underline (_) character	Example: SELECT * FROM MyTable WHERE Name LIKE '%Er_n%';
To begin a single-line comment	Two dashes (--)	-- the following selects everything in my table: SELECT * FROM MyTable;

To bracket a multi-line comment	/* and */	All text between the comment start /* and the comment end */ is ignored.
		
		`/* the following selects everything in my table,`
		`   which we'll then display on the screen */`
		`SELECT * FROM MyTable;`

Entering Multi-line Commands

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

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

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

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

Running SQL Statements From a File

You can also create a file that contains a collection of SQL statements, and then use the `run` command to run those statements. For example:

```
splice> run 'path/to/file.sql';
```

Including Comments

You can include comments on the command line or in a SQL statement file by prefacing the command with two dashes (`--`). Any text following the dashes is ignored by the SQL parser. For example:

```
splice> select * from myTable   -- This selects everything in myTable;
```

Misaligned Quotes

If you mistakenly enter a command line that has misaligned quotation symbols, the interpreter can seem unresponsive. The solution is to add the missing quotation mark(s), followed by a semicolon, and resubmit the line. It won't work as expected, but it will enable you to keep working.

Using rlWrap on the Command Line

rlWrap is a Unix utility that Splice Machine encourages you to use: it allows you to scroll through your command line history, reuse and alter lines, and more. We've included a synopsis of it here.

Example Command Lines

Here are several example command lines:

Operation	Command Example
Display a list of all tables and their schemas	`splice> show tables;`
Display the columns and attributes of a table	`splice> describe tableA;`

Operation	Command Example
Limit the number of rows returned from a select statement	`splice> select * from tableA { limit 10 };`
Print a current time stamp	`splice> values current_timestamp;`

> **NOTE:** Remember that you must end your command lines with the semicolon (`;`) character, which submits the command line to the interpreter.

Scripting splice> Commands

You can use the Splice Machine Command Line Interface (`splice>`) to interactively run database commands. This topic describes how to create a script of `splice>` commands to run a series of operations, such as loading a number of files into your database. To script a series of `splice>` commands, you need to create:

- an SQL commands file containing the SQL statements you want executed
- an SQL file to connect to the database and invoke the SQL commands file
- a shell script using Bash (`/bin/bash`)

Follow these steps to create your script:

1. **Create a file of SQL commands:**
   First, create a file that contains the SQL commands you want to run against your Splice Machine database. For this example, we'll create a file named `create-my-tables.sql` that creates a table in the database:

   ```
   create table customers (
      CUSTOMER_ID BIGINT,
      FIRST_NAME VARCHAR(30),
      LAST_NAME VARCHAR(30)
   );
   ```

2. **Create an SQL file to connect to the database and invoke the commands file**
   We need a separate SQL file named `my_load_datascript.sql`that connects to your database and then invokes the file of SQL commands we just created.

   > **NOTE:** The `connect` command in this file must run before running the file of SQL statements.

   Here we name the first SQL file, and define it to run the SQL statements file named `create-my-tables.sql`:

```
   --First connect to the database
connect 'jdbc:splice://<regionServer>:1527/splicedb';

   --Next run your sql file
run '/users/yourname/create-my-tables.sql';

show tables;
quit;
```

If you are running Splice Machine on a cluster, connect from a machine that is NOT running an HBase RegionServer and specify the IP address of a `regionServer` node, e.g. `10.1.1.110`. If you're using the standalone version of Splice Machine, specify `localhost` instead.

3. **Create a shell script to run your SQL connect file**
   We now create a shell script named `load_datascript.sh` to run the `my_load_datascript.sql` file:

```
#!/bin/bash

export CLASSPATH=<FULL_PATH_TO_SPLICEMACHINE_JAR_FILE>
java -Djdbc.drivers=com.splicemachine.db.jdbc.ClientDriver -Dij.outfile=m
y_load_datascript.out com.splicemachine.db.tools.ij < my_load_datascrip
t.sql
```

   The first line of this script must set the `CLASSPATH` to the location of your Splice Machine jar file. The second line runs the ij command, specifying its output file (`my_load_datascript.out`) and its SQL commands input file, which is the `my_load_datascript_sql` file that we created in the previous step.

4. **Make your shell script executable**
   We need to make the shell script executable with the chmod shell command:

```
chmod +x load_datascript.sh
```

5. **Use nohup to run the script**
   The `nohup` utility allows you to run a script file in the background, which means that it will continue running if you log out, disconnect from a remote machine, or lose your network connection.

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

   Here's the syntax for the `nohup` utility:

```
nohup ./command-name.sh > ./command-name.out 2>&1 &
```

   command-name.sh
   : The name of the shell script or a command name.

   command-name.out
   : The name of the file to capture any output written to `stdout`.

2>&1
: This causes `stderr` (file descriptor `2`) to be written to `stdout` (file descriptor `1`); this means that all output will be captured in `command-name.out`.

&
: The `nohup` utility does not automatically run its command in the background, so we add the `&` to do

Analyze Command

The `analyze` command collects statistics for a specific table, or for an entire schema.

> **NOTE:** Once statistics have been collected for a schema or table, they are automatically used by the query optimizer.

Syntax

```
ANALYZE TABLE [schemaName '.'] table-Name
              [ESTIMATE STATISTICS SAMPLE samplepercent PERCENT];
ANALYZE SCHEMA schema-Name;
```

table-Name
: The name of the table you want to analyze, which can optionally be qualified by its schema name. If you don't specify a *schemaName*, the current schema is assumed.

  You must have insert permission for the table to be able to run this command.

schema-Name
: The name of the schema you want to analyze.

  You must have insert permission for all tables in the schema to be able to run this command.

samplepercent
: A value between `0` and `100` that specifies the sampling percentage to use when generating statistics for this table.

  If you include this clause, statistics are generated by sampling the specified sampling percentage of the table. This can significantly reduce the overhead associated with generating statistics.

  If you do not include this clause, statistics are generated based on the full table.

  .

Analyze Table

The `ANALYZE TABLE` command collects statistics for a specific table in the current schema. It also collects statistics for the index associated with the table in the schema. For example, if you have the following in your database:

- a table named `myTable`
- `myTable` has two indices: `myTableIndex1` and `myTableIndex2`

Then `ANALYZE TABLE` will collect statistics for `myTable`, `myTableIndex1`, and `myTableIndex2`.

The `ANALYZE TABLE` command displays the following information for each partition of the table:

<table>
<tr><th>Value</th><th>Description</th></tr>
<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:

<table><tr><td>0</td><td>Full table (not sampled) statistics that reflect the unmerged partition values.</td></tr><tr><td>1</td><td>Sampled statistics that reflect the unmerged partition values.</td></tr><tr><td>2</td><td>Full table (not sampled) statistics that reflect the table values after all partitions have been merged.</td></tr><tr><td>3</td><td>Sampled statistics that reflect the table values after all partitions have been merged.</td></tr></table></td></tr>
<tr><td>sampleFraction</td><td>The sampling percentage, expressed as 0.0 to 1.0,

» If statsType=0 or statsType=1 (full statistics), this value is not used, and is shown as 0.

» If statsType=2 or statsType=3, this value is the percentage or rows to be sampled. A value of 0 means no rows, and a value of 1 means all rows (same as full statistics).</td></tr>
</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

```
splice> analyze table test.t2;
schemaName |tableName |partition |rowsCollec&|partitionSize |partitionCount  |statsT
ype  |sampleFraction
------------------------------------------------------------------------------------
-----------------------------
TEST       |T2        |-All-     |39226      |235356        |1
|2          |0

1 rows selected
splice>splice> analyze table test.t2 estimate statistics sample 50 percent;
schemaName |tableName |partition |rowsCollec&|partitionSize |partitionCount  |statsT
ype  |sampleFraction
------------------------------------------------------------------------------------
-----------------------------
TEST       |T2        |-All-     |19613      |235356        |1
|3          |0.5

1 rows selected
splice>splice> analyze schema test;
schemaName |tableName |partition |rowsCollec&|partitionSize |partitionCount  |statsT
ype  |sampleFraction
------------------------------------------------------------------------------------
-----------------------------
TEST       |T2        |-All-     |39226      |235356        |1
|2          |0
TEST       |T5        |-All-     |39226      |235356        |1
|2          |0
2 rows selected
splice>
```

Autocommit Command

The `autocommit` command enables or disables auto-commit mode.

JDBC specifies that the default auto-commit mode is enabled; however, certain types of processing require that auto-commit mode be disabled.

Syntax

```
AUTOCOMMIT {ON | OFF}
```

ON
: Enables auto-commit mode.

  If auto-commit mode is changed from disabled (`off`) to enabled (`on`) when there is a transaction outstanding, that work is committed when the current transaction commits, not at the time auto-commit is enabled. Thus, if you are enabling auto-commit when a transaction is outstanding, first use either the Rollback command to ensure that all prior work is completed before the return to auto-commit mode.

OFF
: Disables auto-commit mode.

Examples

```
splice> autocommit off;
splice> DROP TABLE menu;
0 rows inserted/updated/deleted
splice> CREATE TABLE menu (course CHAR(10), item CHAR(20), price INT);
0 rows inserted/updated/deleted
splice> INSERT INTO menu VALUES ('entree', 'lamb chop', 14),
('dessert', 'creme brulee', 6),
('appetizer', 'baby greens', 7);
3 rows inserted/updated/deleted
splice> commit;
splice> autocommit on;
splice>
```

Commit Command

The `commit` command issues a `java.sql.Connection.commit` request, which commits the currently active transaction and initiates a new transaction.

> **NOTE:** You should only use this command when auto-commit mode is disabled.

Syntax

```
COMMIT
```

Examples

```
splice> commit;
splice>
```

Execute Command

The `execute` command executes an SQL command string or a prepared statement.

Syntax

```
EXECUTE { SQLString | PreparedStatementIdentifier }
          [ USING { String | Identifier } ]
```

SQLString
: The SQL command string to execute; this string is passed to the connection without further processing by the command line interpreter.

PreparedStatementIdentifier
: The identifier of the prepared statement to execute; this must be the name associated with a prepared statement when created by the Prepare command.

String
: Use this or *Identifier* to supply values for dynamic parameters, if the command being executed contains them.

Identifier
: Use this or *String* to supply values for dynamic parameters, if the command being executed contains them. This identifier must have a result set as its result:

- Each row of the result set is applied to the input parameters of the command to be executed, so the number of columns in the `Using` clause's result set must match the number of input parameters in the statement being executed.
- The command line interpreter displays the results of each execution of the statement as they are created.
- If the Using clause's result set contains zero rows, the statement is not executed.

Examples

```
splice> autocommit off;
splice> prepare menuInsert as 'INSERT INTO menu VALUES (?, ?, ?)';
splice> execute menuInsert using 'VALUES
(''entree'', ''lamb chop'', 14),
(''dessert'', ''creme brulee'', 6)';
1 row inserted/updated/deleted
1 row inserted/updated/deleted
splice> commit;
splice> connect 'jdbc:splice://abc:1527/splicedb;user=me;password=mypswd';
splice> create table firsttable (id int primary key,
name varchar(12));
0 rows inserted/updated/deleted
splice> insert into firsttable values
10,'TEN'),(20,'TWENTY'),(30,'THIRTY');
3 rows inserted/updated/deleted
splice> select * from firsttable;
ID         |NAME
------------------------
10         |TEN
20         |TWENTY
30         |THIRTY

3 rows selected
splice> connect 'jdbc:splice://xyz:1527/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.

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

Value	Description
`true`	The exported files are compressed using `deflate/gzip`.
`false`	Exported files are not compressed.

replicationCount
: The file system block replication count to use for the exported CSV files.

  You can specify any positive integer value. The default value is `1`.

fileEncoding
: The character set encoding to use for the exported CSV files.

  You can specify any character set encoding that is supported by the Java Virtual Machine (JVM). The default encoding is `UTF-8`.

fieldSeparator
: The character to use for separating fields in the exported CSV files.

  The default separator character is the comma (`,`).

quoteCharacter
: The character to use for quoting output in the exported CSV files.

  The default quote character is the double quotation mark (`"`).

Usage

The `EXPORT` command generates one or more CSV files and stores them in the directory that you specified in the `exportPath` parameter. More than one output file is generated to enhance the parallelism and performance of this operation.

If `compression=true`, then each of the generated files is named with this format:

```
export_<N>.csv.gz
```

If `compression=false`, then each of the generated files is named with this format:

```
export_<N>.csv
```

The value of `<N>` is a random integer value.

Merging the Exported Files

You can copy all of the exported files into a single file on your local file system using the Hadoop FS command `getmerge`. The syntax for `getmerge` is:

```
hadoop fs -getmerge sourceDir localPath
```

Use the *exportPath* directory as the value of sourceDir to copy all of the exported CSV files to your *localPath*.

For more information about the `getmerge` command, see http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html#getmerge.

Examples

```
        -- This example uses all default options:
 splice> EXPORT('/my/export/dir', false, null, null, null, null)
          SELECT a,b,sqrt(c) FROM t1 join t2 on t1.a=t2.a;

        -- This example explicitly specifies options:
splice> EXPORT('/my/export/dir', false, 3, 'utf-8', '|', ';')
          SELECT a,b,sqrt(c) FROM t1 join t2 on t1.a=t2.a;
```

Prepare Command

The `prepare` command creates a `java.sql.PreparedStatement` using the value of the specified SQL command *String*, and assigns an identifier to the prepared statement so that other `splice>` commands can use the statement.

If a prepared statement with the specified *Identifier* name already exists in the command interpreter, an error is returned, and the previous prepared statement is left unchanged. If there are any errors in preparing the statement, no prepared statement is created.

If the *Identifier* specifies a connection Name, the statement is prepared on the specified connection.

Syntax

```
PREPARE Identifier AS String
```

Identifier
: The identifier to assign to the prepared statement.

String
: The command string to prepare.

Examples

```
splice> prepare seeMenu as 'SELECT * FROM menu';
splice> execute seeMenu;
COURSE    |ITEM                |PRICE
-------------------------------------------
entree    |lamb chop           |14
dessert   |creme brulee        |6

splice> prepare addYears as 'update children set age = age + ? where name = ?';
splice> execute addYears using 'values (10, ''Abigail'')';
```

Release Savepoint Command

The `release savepoint` command issues a `java.sql.Connection.releaseSavepoint` request, which releases a savepoint within the current transaction. Once a savepoint has been released, attempting to reference it in a rollback operation will cause an `SQLException` to be thrown.

When you commit a transaction, any savepoints created in that transaction are automatically released and invalidated when the transaction is committed or the entire transaction is rolled back.

> **NOTE:** When you rollback a transaction to a savepoint, that savepoint and any others created after it within the transaction are automatically released.

Syntax

```
release savepoint identifier;
```

identifier
: The name of the savepoint to release.

Examples

Example

First we'll create a table, turn `autocommit` off, and insert some data into the table. We then create a savepoint, and verify the contents of our table:

```
splice> CREATE TABLE myTbl(i int);
0 rows inserted/updated/deleted
splice> AUTOCOMMIT OFF;
splice> INSERT INTO myTbl VALUES 1,2,3;
3 rows inserted/updated/deleted
splice> SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----------
1
2
3

3 rows selected
```

Next we add new values to the table and again verify its contents:

```
splice> INSERT INTO myTbl VALUES 4,5;
2 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----------
1
2
3
4
5

5 rows selected
```

Now we release our original savepoint, insert a few more values, and create a new savepoint, `savept2`.

```
splice> RELEASE SAVEPOINT savept1;
0 rows inserted/updated/deletedsplice> 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 trans
action.
splice> ROLLBACK TO SAVEPOINT savept2;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----------
1
2
3
4
5
6
7

7 rows selected
```

And finally, we commit the transaction:

```
COMMIT;
```

See Also

- savepoint command
- rollback to savepoint command
- The *Running Transactions* topic contains includes a discussion of using savepoints.

Remove Command

The `remove` command removes a previously prepared statement from the command line interpreter.

The statement is closed, releasing its database resources.

Syntax

```
REMOVE Identifier
```

Identifier
The name assigned to the prepared statement when it was prepared with the Prepare statement.

Examples

```
splice> prepare seeMenu as 'SELECT * FROM menu';
splice> execute seeMenu;
COURSE    |ITEM                |PRICE
-----------------------------------------------
entree    |lamb chop           |14
dessert   |creme brulee        |6

2 rows selected
splice> remove seeMenu;
splice> execute seeMenu;
splice ERROR: Unable to establish prepared statement SEEMENU
splice>
```

Rollback Command

The `rollback` command issues a `java.sql.Connection.rollback` request, which rolls back (undoes) the currently active transaction and initiates a new transaction.

> **NOTE:** You should only use this command when auto-commit mode is disabled.

Usage Notes

In contrast to the `Rollback to Savepoint` command, the `Rollback` command aborts the current transaction and starts a new one.

Examples

```
splice> autocommit off;
splice> INSERT INTO menu VALUES ('dessert', 'rhubarb pie', 4);
1 row inserted/updated/deleted
splice> SELECT * from menu;
COURSE    |ITEM                |PRICE
-----------------------------------------------
entree    |lamb chop           |14
dessert   |creme brulee        |7
appetizer |baby greens         |7
dessert   |rhubarb pie         |4

4 rows selected
splice> rollback;
splice> SELECT * FROM menu;
COURSE    |ITEM                |PRICE
-----------------------------------------------
entree    |lamb chop           |14
dessert   |creme brulee        |7
appetizer |baby greens         |7

3 rows selected
splice>
```

Rollback to Savepoint Command

The `rollback to savepoint` command issues a `java.sql.Connection.rollback` request, which has been overloaded to work with a savepoint within the current transaction.

> **NOTE:** When you rollback a transaction to a savepoint, that savepoint and any others created after it within the transaction are automatically released.

Syntax

```
rollback to savepoint identifier;
```

identifier

The name of the savepoint to which the transaction should be rolled back: all savepoints up to and including this one are rolled back.

Usage Notes

In contrast to the `Rollback` command, the `Rollback to Savepoint` command rolls back but of your work, but does not start a new transaction.

Examples

First we'll create a table, turn autocommit off, and insert some data into the table. We then create a savepoint, and verify the contents of our table:

```
splice> CREATE TABLE myTbl(i int);
0 rows inserted/updated/deleted
splice> AUTOCOMMIT OFF;
splice> INSERT INTO myTbl VALUES 1,2,3;
3 rows inserted/updated/deleted
splice> SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----------
1
2
3

3 rows selected
```

Next we add new values to the table and again verify its contents:

```
splice> INSERT INTO myTbl VALUES 4,5;
2 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----------
1
2
3
4
5
5 rows selected
```

Now we roll back to our savepoint, and verify that the rollback worked:

```
splice> ROLLBACK TO SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----------
1
2
3

3 rows selected
```

And finally, we commit the transaction:

```
COMMIT;
```

See Also

- savepoint command
- release savepoint command
- The *Running Transactions* topic contains includes a discussion of using savepoints.

Savepoint Command

The `savepoint` command issues a `java.sql.Connection.setSavepoint` request, which sets a savepoint within the current transaction.

Savepoints are only useful when autocommit is off.

> **NOTE:** You can define multiple savepoints within a transaction.

Syntax

```
savepoint identifier;
```

identifier
An identifier name for the string.

Example

Example

First we'll create a table, turn `autocommit` off, and insert some data into the table. We then create a savepoint, and verify the contents of our table:

```
splice> CREATE TABLE myTbl(i int);
0 rows inserted/updated/deleted
splice> AUTOCOMMIT OFF;
splice> INSERT INTO myTbl VALUES 1,2,3;
3 rows inserted/updated/deleted
splice> SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----------
1
2
3

3 rows selected
```

Next we add new values to the table and again verify its contents:

```
splice> INSERT INTO myTbl VALUES 4,5;
2 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----------
1
2
3
4
5

5 rows selected
```

Now we roll back to our savepoint, and verify that the rollback worked:

```
splice> ROLLBACK TO SAVEPOINT savept1;
0 rows inserted/updated/deleted
splice> SELECT * FROM myTbl;
I
-----------
1
2
3

3 rows selected
```

And finally, we commit the transaction:

```
COMMIT;
```

See Also

- release savepoint command
- rollback to savepoint command
- The *Running Transactions* topic contains includes a discussion of using savepoints.

Connect Command

The `connect` command connects to the database specified by `ConnectionURLString`. It connects by issuing a `getConnection` request with the specified URL, using `java.sql.DriverManager` or `javax.sql.DataSource` to set the current connection to that URL.

Syntax

```
CONNECT ConnectionURLString  [ AS Identifier ]
```

ConnectionURLString
: The URL of the database. Note that this URL typically includes connection parameters such as user name, password, or security options.

Identifier
: The optional name that you want to assign to the connection.

If the connection succeeds, the connection becomes the current one and all further commands are processed against the new, current connection.

Example 1: Connecting on a Cluster

If you are running Splice Machine on a cluster, connect from a machine that is NOT running an HBase RegionServer and specify the IP address of a `regionServer` node, e.g. `10.1.1.110`.

```
splice> connect 'jdbc:splice://regionServer:1527/splicedb';
```

This example includes a user ID and password in the connection URL string:

```
splice> connect 'jdbc:splice://1.2.3.456:1527/splicedb;user=splice;password=admin';
```

And this example includes specifies that SSL peer authentication will be enabled for the connection:

```
splice> connect 'jdbc:splice://1.2.3.456:1527/splicedb;user=splice;password=admin;ss
l=peerAuthentication';
```

NOTE: You can only connect with SSL/TLS if your administrator has configured this capability for you.

Example 2: Connecting to the Standalone Version

If you're using the standalone version of Splice Machine, specify `localhost` instead.

```
splice> connect 'jdbc:splice://localhost:1527/splicedb';
```

Here is an example that includes a user ID and password in the connect string:

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

Column Name	Description
`COLUMN_NAME`	The name of the column
`TYPE_NAME`	The data type of the column
`DECIMAL_DIGITS`	The number of fractional digits
`NUM_PREC_RADIX`	The radix, which is typically either 10 or 2
`COLUMN_SIZE`	The column size: » For char or date types, this is the maximum number of characters » For numeric or decimal types, this is the precision
`COLUMN_DEF`	The default value for the column
`CHAR_OCTE`	Maximum number of bytes in the column
`IS_NULL`	Whether (YES) or not (NO) the column can contain null values

Examples

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

5 rows selected
```

Disconnect Command

The `disconnect` command disconnects from a database. It issues a `java.sql.Connection.close` request for the current connection, or for the connection(s) specified on the command line.

Note that disconnecting from a database does not stop the command line interface or shut down Splice Machine. You can use the exit command to close out of the command line interface.

Syntax

```
DISCONNECT [ALL | CURRENT | connectionIdentifier]
```

ALL
: All known connections are closed; as a result, there will not be a current connection.

CURRENT
: The current connection is closed. This is the default behavior.

connectionIdentifier
: The name of the connection to close; this must be same identifier assigned when the connection was opened with a Connect command.

Examples

```
splice> connect 'jdbc:splice://xyz:1527/splicedb';
splice> -- we create a new table in splicedb:
CREATE TABLE menu(course CHAR(10), ITEM char(20), PRICE integer);
0 rows inserted/updated/deleted
splice> disconnect;
splice>
```

ElapsedTime Command

The `elapsedtime` command enables or disables having the command line interface display the amount of time required for a command to complete its execution.

Syntax

```
ELAPSEDTIME { ON | OFF }
```

ON
: Enables display of the elapsed time by the command line interface. When this is enabled, you'll see how much time elapsed during execution of the command line.

OFF
: Disables display of the elapsed time.

Examples

```
splice> elapsedtime on;
splice> VALUES current_date;
1
----------
2014-06-24
ELAPSED TIME = 2134 milliseconds
splice>
```

Exit Command

The `exit` or `quit` command causes the command line interface to exit. Issuing this command from within a command file causes the outermost input loop to halt.

Syntax

```
EXIT
```

Examples

```
splice> exit;
```

Help Command

The `help` command displays a list of the available `splice>` commands.

Syntax

```
HELP
```

Example

```
splice> help;

 Supported commands include:

  CONNECT 'url for database' [ PROTOCOL namedProtocol ] [ AS connectionName ];
                              -- connects to database URL
                              -- and may assign identifier
  AUTOCOMMIT [ ON | OFF ];    -- sets autocommit mode for the connection
  SHOW SCHEMAS;               -- lists all schemas in the current database
  SHOW [ TABLES | VIEWS | PROCEDURES | FUNCTIONS | SYNONYMS] { IN schema };
                              -- lists tables, views, procedures, functions or 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 direct
ly.

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 intepreter.

  Set this value to `0` to display the entire content of each column.

Examples

```
splice> maximumdisplaywidth 3;
splice> VALUES 'NOW IS THE TIME!';
1
---
NOW
splice> maximumdisplaywidth 30;
splice> VALUES 'NOW IS THE TIME!';
1
----------------
NOW IS THE TIME!
```

Run Command

The `run` command redirects the command line interpreter to read and process commands from the specified file. This continues until the end of the file is reached, or an exit command is executed. Note that the file *can* contain `run` commands.

> **NOTE:** You can specify a file that is in the directory where you are running the command, or you can include the full file path so that the `run` command can find it.

The command line interpreter prints out the statements in the file as it executes them.

Syntax

```
RUN String
```

String
The name of the file containing commands to execute.

Examples

```
splice> run 'setupMenuConn.spl';
splice> -- this is setupMenuConn.spl
-- splice displays its contents as it processes file
splice> connect 'jdbc:splice://xyz:1527/splicedb';
splice> autocommit off;
splice> -- this is the end of setupMenuConn.spl
-- there is now a connection to splicedb on xyz and no autocommit.
-- input will now resume from the previous source.
;
splice>
```

Set Connection Command

The `set connection` command allows you to specify which connection is the current connection, when you have more than one connection open.

> **NOTE:** If the specified connection does not exist, an error results, and the current connection is unchanged.

Syntax

```
SET CONNECTION Identifier
```

Identifier

The name of the connection that you want to be the current connection.

Examples

```
splice> connect 'jdbc:splice://abc:1527/splicedb;user=splice;password=admin' as samp
le1;
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>
```

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 Functions

The `show functions` command displays all of the functions in the database, or the names of the functions in the specified schema.

Syntax

```
SHOW FUNCTIONS [ IN schemaName ]
```

schemaName
: If you supply a schema name, only the functions in that schema are displayed; otherwise, all functions in the database are displayed.

Results

The `show functions` command results contains the following columns:

Column Name	Description
FUNCTION_SCHEMA	The name of the function's schema
FUNCTION_NAME	The name of the function
REMARKS	Any remarks that have been stored for the function

Examples

```
splice> CREATE FUNCTION TO_DEGREES ( RADIANS DOUBLE )
> RETURNS DOUBLE
> PARAMETER STYLE JAVA
> NO SQL LANGUAGE JAVA
> EXTERNAL NAME 'java.lang.Math.toDegrees';
0 rows inserted/updated/deleted
splice> show functions in splice;
FUNCTION_SCHEM|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:

Column Name	Description
TABLE_NAME	The name of the table.
INDEX_NAME	The name of the index.
COLUMN_NAME	The name of the column.
ORDINAL	The position of the column in the index.
NON_UNIQUE	Whether this is a unique or non-unique index.
TYPE	The index type
ASC_	Indicates if this is an ascending (A) or descending (D) index.
CONGLOM_NO	The conglomerate number, which points to the corresponding table in HBase.

Examples

```
splice> show indexes from my_table;
TABLE_NAME    |INDEX_NAME       |COLUMN_NAME    |ORDINAL&|NON_UNIQUE|TYPE |ASC&|CONGL
OM_NO
-------------------------------------------------------------------------------------
-----
MY_TABLE      |I1               |ID             |1       |true      |BTREE|A   |1937
MY_TABLE      |I1               |STATE_CD       |2       |true      |BTREE|A   |1937
MY_TABLE      |I1               |CITY           |3       |true      |BTREE|A   |1937
MY_TABLE      |I2               |ID             |1       |true      |BTREE|D   |1953
MY_TABLE      |I2               |STATE_CD       |2       |true      |BTREE|D   |1953
MY_TABLE      |I2               |CITY           |3       |true      |BTREE|D   |1953
MY_TABLE      |I3               |ID             |1       |true      |BTREE|D   |1969
MY_TABLE      |I3               |STATE_CD       |2       |true      |BTREE|A   |1969
MY_TABLE      |I3               |CITY           |3       |true      |BTREE|D   |1969
MY_TABLE      |I4               |LATITUDE       |1       |true      |BTREE|D   |1985
MY_TABLE      |I4               |STATE_CD       |2       |true      |BTREE|A   |1985
MY_TABLE      |I4               |ID             |3       |true      |BTREE|A   |1985
MY_TABLE      |I5               |ID             |1       |true      |BTREE|A   |2001
MY_TABLE      |I5               |ID             |2       |true      |BTREE|D   |2001

14 rows selected
splice>
```

Show PrimaryKeys

The `show primarykeys` command displays all the primary keys in the specified table.

Syntax

```
SHOW PRIMARYKEYS FROM schemaName.tableName
```

schemaName
The schema name.

tableName
The table name.

Results

The `show primary keys` command results contains the following columns:

Column Name	Description
TABLE_NAME	The name of the table
COLUMN_NAME	The name of the column
KEY_SEQ	The order of the column within the primary key
PK_NAME	The unique name of the constraint

Examples

```
splice> create table myTable(i int, j int, primary key (i,j));
0 rows inserted/updated/deleted

splice> show primarykeys from mySchema.myTable;
TABLE_NAME  |COLUMN_NAME  |KEY_SEQ  |PK_NAME
------------------------------------------------------
A           |I            |1        |SQL141120202723310
A           |J            |2        |SQL141120202723310

2 rows selected
```

Show Procedures

The `show procedures` command displays all of the procedures that have been created with the `create procedure` statement in the database or specified schema.

Syntax

```
SHOW PROCEDURES [ IN schemaName ]
```

schemaName

If you supply a schema name, only the procedures in that schema are displayed; otherwise, all procedures in the database are displayed.

Results

The `show procedures` command results contains the following columns:

Column Name	Description
`PROCEDURE_SCHEMA`	The name of the procedure's schema
`PROCEDURE_NAME`	The name of the procedure
`REMARKS`	Any remarks that have been stored for the procedure

Examples

```
splice> show procedures in syscs_util;
PROCEDURE_SCHEM |PROCEDURE_NAME                              |REMARKS
------------------------------------------------------------------------------SYS
CS_UTIL      |COLLECT_SCHEMA_STATISTICS                |com.splicemachine. ...
SYSCS_UTIL      |COLLECT_TABLE_STATISTICS                |com.splicemachine. ...
SYSCS_UTIL      |DISABLE_COLUMN_STATISTICS               |com.splicemachine. ...
SYSCS_UTIL      |DROP_SCHEMA_STATISTICS                  |com.splicemachine. ...
SYSCS_UTIL      |DROP_TABLE_STATISTICS                   |com.splicemachine. ...
SYSCS_UTIL      |ENABLE_COLUMN_STATISTICS                |com.splicemachine. ...
SYSCS_UTIL      |GET_ACTIVATION                          |com.splicemachine. ...
SYSCS_UTIL      |IMPORT_DATA                             |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_BACKUP_DATABASE                   |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_CANCEL_BACKUP                     |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_CANCEL_DAILY_BACKUP               |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_COMMIT_CHILD_TRANSACTION          |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_CREATE_USER                       |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_DELETE_BACKUP                     |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_DELETE_OLD_BACKUPS                |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_DROP_USER                         |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_ELEVATE_TRANSACTION               |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_EMPTY_GLOBAL_STATEMENT_CACHE      |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_EMPTY_STATEMENT_CACHE             |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_ENABLE_ENTERPRISE                 |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_FLUSH_TABLE                       |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_ACTIVE_SERVERS                |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_ACTIVE_TRANSACTION_IDS        |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_ALL_PROPERTIES                |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_AUTO_INCREMENT_ROW_LOCATIONS  |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_CURRENT_TRANSACTION           |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_GLOBAL_DATABASE_PROPERTY      |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_LOGGERS                       |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_LOGGER_LEVEL                  |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_REGION_SERVER_CONFIG_INFO     |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_REGION_SERVER_STATS_INFO      |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_REQUESTS                      |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_SCHEMA_INFO                   |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_STORED_STATEMENT_PLAN_INFO    |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_TIMESTAMP_GENERATOR_INFO      |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_TIMESTAMP_REQUEST_INFO        |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_VERSION_INFO                  |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_GET_WRITE_INTAKE_INFO             |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_INVALIDATE_STORED_STATEMENTS      |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_KILL_STALE_TRANSACTIONS           |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_KILL_TRANSACTION                  |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_MODIFY_PASSWORD                   |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_PERFORM_MAJOR_COMPACTION_ON_SCHEMA |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_PERFORM_MAJOR_COMPACTION_ON_TABLE  |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_RECOMPILE_INVALID_STORED_STATEMENTS|com.splicemachine. ...
SYSCS_UTIL      |SYSCS_REFRESH_EXTERNAL_TABLE            |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_RELOAD_SECURITY_POLICY            |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_RESET_PASSWORD                    |com.splicemachine. ...
```

```
SYSCS_UTIL      |SYSCS_RESTORE_DATABASE                    |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_RESTORE_DATABASE_OWNER              |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_SCHEDULE_DAILY_BACKUP               |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_SET_GLOBAL_DATABASE_PROPERTY        |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_SET_LOGGER_LEVEL                    |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_SET_USER_ACCESS                     |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_SPLIT_REGION_AT_POINTS              |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_SPLIT_TABLE                         |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_SPLIT_TABLE_AT_POINTS               |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_SPLIT_TABLE_EVENLY                  |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_START_CHILD_TRANSACTION             |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_UPDATE_ALL_SYSTEM_PROCEDURES        |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_UPDATE_METADATA_STORED_STATEMENTS   |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_UPDATE_SCHEMA_OWNER                 |com.splicemachine. ...
SYSCS_UTIL      |SYSCS_UPDATE_SYSTEM_PROCEDURE             |com.splicemachine. ...
SYSCS_UTIL      |UPSERT_DATA_FROM_FILE                     |com.splicemachine. ...
SYSCS_UTIL      |VACUUM                                    |com.splicemachine. ...
66 rows selected
```

Show Roles

The `show roles` command a sorted list of all of the roles defined in the database.

Syntax

```
SHOW ROLES
```

Examples

```
splice> create role testRole;
0 rows inserted/updated/deleted
splice> show roles;

ROLEID
------------------------------
TESTROLE

1 row selected
```

Show Schemas

The `show schemas` command displays all of the schemas in the current connection.

Syntax

```
SHOW SCHEMAS
```

Examples

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

Show Synonyms

The `show synonyms` command displays all of the synonyms that have been created with the `CREATE SYNONYM` statement in the database or specified schema.

Syntax

```
SHOW SYNONYMS [ IN schemaName ]
```

schemaName
: If you supply a schema name, only the synonyms in that schema are displayed; otherwise, all synonyms in the database are displayed.

Results

The `show synonyms` command results contains the following columns:

Column Name	Description
TABLE_SCHEMA	The name of the synonym's schema
TABLE_NAME	The name of the table
CONGLOM_ID	The conglomerate number, which points to the corresponding table in HBase
REMARKS	Any remarks associated with the table

Examples

```
splice> show synonyms;
TABLE_SCHEM  |TABLE_NAME         |CONGLOM_ID|REMARKS
------------------------------------------------------------------
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:

Column Name	Description
`TABLE_SCHEMA`	The name of the table's schema
`TABLE_NAME`	The name of the table
`CONGLOM_ID`	The conglomerate number, which points to the corresponding table in HBase
`REMARKS`	Any remarks associated with the table

Examples

```
splice> show tables;
TABLE_SCHEM  |TABLE_NAME          |CONGLOM_ID|REMARKS
------------------------------------------------------
SYS          |SYSALIASES          |352       |
SYS          |SYSBACKUP           |1040      |
SYS          |SYSBACKUPFILESET    |1056      |
SYS          |SYSBACKUPITEMS      |1136      |
SYS          |SYSBACKUPJOBS       |1200      |
SYS          |SYSCHECKS           |368       |
SYS          |SYSCOLPERMS         |704       |
SYS          |SYSCOLUMNS          |64        |
SYS          |SYSCOLUMNSTATS      |1216      |
SYS          |SYSCONGLOMERATES    |80        |
SYS          |SYSCONSTRAINTS      |320       |
SYS          |SYSDEPENDS          |240       |
SYS          |SYSFILES            |256       |
SYS          |SYSFOREIGNKEYS      |288       |
SYS          |SYSKEYS             |272       |
SYS          |SYSPERMS            |784       |
SYS          |SYSPHYSICALSTATS    |1232      |
SYS          |SYSPRIMARYKEYS      |384       |
SYS          |SYSROLES            |736       |
SYS          |SYSROUTINEPERMS     |720       |
SYS          |SYSSCHEMAPERMS      |1392      |
SYS          |SYSSCHEMAS          |32        |
SYS          |SYSSEQUENCES        |752       |
SYS          |SYSSTATEMENTS       |304       |
SYS          |SYSTABLEPERMS       |688       |
SYS          |SYSTABLES           |48        |
SYS          |SYSTABLESTATS       |1248      |
SYS          |SYSTRIGGERS         |656       |
SYS          |SYSUSERS            |816       |
SYS          |SYSVIEWS            |336       |
SYSIBM       |SYSDUMMY1           |1344      |
SPLICE       |MYTABLE             |1536      |

32 rows selected

splice>show tables in SPLICE;
TABLE_SCHEM  |TABLE_NAME          |CONGLOM_ID|REMARKS
------------------------------------------------------
SPLICE       |MYTABLE             |1536      |

1 row selected
```

Show Views

The `show views` command displays all of the views in the current or specified schema.

Syntax

```
SHOW VIEWS [ IN schemaName ]
```

schemaName
: If you supply a schema name, only the views in that schema are displayed; otherwise, the views in the current schema are displayed.

Results

The `show views` command results contains the following columns:

Column Name	Description
TABLE_SCHEMA	The name of the table's schema
TABLE_NAME	The name of the table
CONGLOM_ID	The conglomerate number, which points to the corresponding table in HBase
REMARKS	Any remarks associated with the table

Examples

```
splice> show views;
TABLE_SCHEM  |TABLE_NAME          |CONGLOM_ID|REMARKS
------------------------------------------------------------------
SPLICE       |GUITAR_BRANDS       |4321      |
0 rows selected
```