



Developer Tutorials

Last generated: March 01, 2018

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

Table of Contents

Developer Tutorials

Splice Machine Tutorials

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

splice> Command Line

Getting Started With splice>	2
Scripting splice>	15
RIWrap Summary	20

Ingestion and Streaming

Uploading Data to S3	22
Setting up S3 Access	30
Importing Data: Tutorial Overview	38
Importing Data: Parameter Usage	44
Importing Data: Input Data Handling	56
Importing Data: Error Handling	66
Importing Data: Usage Examples	68
Importing Data: Bulk HFile Examples	74
Importing Data: Importing TPCCH Data	85
Creating a Kafka Producer	110
Configuring a Kafka Feed	111
Using Apache Storm	112
MQTT Spark Streaming	116

Analytics and Machine Learning

Using Zeppelin	125
----------------------	-----

Connecting Programmatically

Connecting through HAProxy	130
Connecting with Java	137
Connecting with JRuby	140
Connecting with Jython	143
Connecting with Scala	146
Connecting with NodeJS	150
Installing our ODBC Driver	151
Connecting with Python	166

Connecting with C and ODBC	169
Connecting BI Tools	
Connecting DBeaver	171
Connecting DBVisualizer	179
Connecting Cognos	183
Connecting SquirrelL	184
Connecting Tableau	192

Splice Machine Tutorials

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

Section	Description
Data Ingestion and Streaming	<p>This section includes topics that show you how to get data into your database:</p> <ul style="list-style-type: none"> • Importing Data Tutorial • Streaming with Kafka and Splice Machine • Spark Streaming with MQTT • Using Apache Storm with Splice Machine
Connecting Your Database	<p>This section introduces our JDBC and ODBC drivers and contains mini-tutorials to help you use our drivers to connect to your Splice Machine database with various programming languages, including:</p> <ul style="list-style-type: none"> • Connecting with AngularJS and JDBC • Connecting with Java and JDBC • Connecting with JRuby and JDBC • Connecting with Jython and JDBC • Connecting with Scala and JDBC • Connecting with C and ODBC • Connecting with Python and ODBC <p>This section also contains mini-tutorials to help you connect specific Business Intelligence tools to your Splice Machine database, including:</p> <ul style="list-style-type: none"> • Connecting Cognos to Splice Machine • Connecting DBeaver to Splice Machine • Connecting DBVisualizer to Splice Machine • Connecting SQuirreL to Splice Machine • Connecting Tableau to Splice Machine
Analytics and Machine Learning	<ul style="list-style-type: none"> • Using Zeppelin with Splice Machine

Getting Started With the splice> Command Line Interface

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

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



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

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

Watch the Video

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

Follow the Written Version

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

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

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

Starting splice>

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

1. Open a terminal window

2. Navigate to your splicemachine directory

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

3. Start splice>

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

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

4. The command line interpreter starts:

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

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

Restarting splice>

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

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

Basic Syntax Rules

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

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

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


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

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

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

The [Command Line Syntax](#) topic contains a complete syntax reference for `splice>`.

Connecting to a Database

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

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

Anatomy of a Connection String

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

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

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

To view your current connections, use the [show connections](#) command:

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

You can use the [set connection](#) command to modify the current connection:

```
splice> SET CONNECTION DEMO;
splice> show connections;
DEMO*     - jdbc:splice://srv55:1527/splicedb
SPLICE    - jdbc:splice://localhost:1527/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_SCHEM
-----
NULLID
SPLICE
SQLJ
SYS
SYSCAT
SYSCS_DIAG
SYSCS_UTIL
SYSFUN
SYSIBM
SYSPROC
SYSSTAT
11 rows selected
```

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

To display the current schema name, use the built-in [current_schema](#) function:

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

To change which schema is current, use the SQL [set_schema](#) statement:

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

Displaying Tables

The [show tables](#) command displays a list of all tables in all of the schemas in your database:

```
splice> SHOW TABLES;
```

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

34 rows selected

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

```
splice> show tables in SPLICE;
```

TABLE_SCHEM	TABLE_NAME	CONGLOM_ID	REMARKS
SPLICE	CUSTOMERS	1568	
SPLICE	T_DETAIL	1552	
SPLICE	T_HEADER	1536	

3 rows selected

To examine the structure of a specific table, use the [DESCRIBE](#) command:

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

5 rows selected
```

Displaying Indexes

You can display all of the indexes in a schema:

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

5 rows selected
```

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

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

3 rows selected
```



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

Displaying Views

Similarly to indexes, you can use the [show views](#) command to display all of the indexes in your database or in a schema:

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

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

2 rows selected
```

Displaying Stored Procedures and Functions

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

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

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

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

3 rows selected
```

Basic DDL and DML Statements

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

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

See the [DML Statements](#) sections in our *SQL Reference Manual* for more information.

CREATE Statements

SQL uses `CREATE` statements to create objects such as [tables](#). For example:

```

splice> CREATE schema MySchema1;
0 rows inserted/updated/deleted
splice> create Schema mySchema2;
0 rows inserted/updated/deleted
splice> show schemas;
TABLE_SCHEM
-----
MYSHEMA1MYSHEMA2NULLID
SPlice
SQLJ
SYS
SYSCAT
SYSCS_DIAG
SYSCS_UTIL
SYSFUN
SYSIBM
SYSPROC
SYSSTAT
13 rows selected
splice> SET SCHEMA MySchema1;
0 rows inserted/updated/deleted
splice> CREATE TABLE myTable ( myNum int, myName VARCHAR(64) );
0 rows inserted/updated/deleted
splice> CREATE TABLE Players(
    ID                SMALLINT NOT NULL PRIMARY KEY,
    Team              VARCHAR(64) NOT NULL,
    Name              VARCHAR(64) NOT NULL,
    Position          CHAR(2),
    DisplayName       VARCHAR(24),
    BirthDate         DATE
);
0 rows inserted/updated/deleted
splice> SHOW TABLES IN MySchema1;
TABLE_SCHEM | TABLE_NAME | CONGLOM_ID | REMARKS
-----
MYSHEMA1    | MYTABLE     | 1616       |
MYSHEMA1    | PLAYERS     | 1632       |
2 rows selected

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

```


See the [CREATE Statements](#) section in our *SQL Reference Manual* for more information.

DROP Statements

SQL uses `DROP` statements to delete objects such as [tables](#). For example:

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

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

```
splice> show schemas;
TABLE_SCHEM
-----
MYSCHEMA1
MYSCHEMA2
NULLID
SPLICE
SQLJ
SYS
SYSCAT
SYSCS_DIAG
SYSCS_UTIL
SYSFUN
SYSIBM
SYSPROC
SYSSTAT
12 rows selected

splice> DROP TABLE myTable;
0 rows inserted/updated/deleted
splice> SHOW TABLES IN MySchema1;
TABLE_SCHEM |TABLE_NAME          |CONGLOM_ID|REMARKS
-----
MYSCHEMA1   |PLAYERS              |1632      |1 row selected
```

See the [DROP Statements](#) section in our *SQL Reference Manual* for more information.

Inserting Data

Once you've created a table, you can use [INSERT](#) statements to insert records into that table; for example:

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

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

Selecting and Displaying Data

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

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

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

3 rows selected
```

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

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

3 rows selected
```

You can also qualify which records to select with a [WHERE](#) clause; for example:

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

You can easily count the records in a table by using the [SELECT](#) statement; for example:

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

See Also

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

Scripting the splice> Command Line Interface

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

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

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

Running a File of splice> Commands

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

1. Create a file of SQL commands:

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

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

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

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

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

SELECT * FROM Players;
```

2. Start splice>

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

3. Run the SQL Script

Now, in `splice>`, run your script with the [run](#) command:

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

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

```
splice> connect 'jdbc:splice://localhost:1527/splicedb;user=splice;password=admin';
splice> create table players (
  ID SMALLINT NOT NULL PRIMARY KEY,
  Team VARCHAR(64) NOT NULL,
  Name VARCHAR(64) NOT NULL,
  Position CHAR(2),
  DisplayName VARCHAR(24),
  BirthDate DATE );
0 rows inserted/updated/deleted
splice> INSERT INTO Players
  VALUES (99, 'Giants', 'Joe Bojangles', 'C', 'Little Joey', '07/11/1991'),
          (73, 'Giants', 'Lester Johns', 'P', 'Big John', '06/09/1984'),
          (27, 'Cards', 'Earl Hastings', 'OF', 'Speedy Earl', '04/22/1982');
3 rows inserted/updated/deleted
splice> SELECT * FROM Players;
ID      |TEAM      |NAME                |POS&|DISPLAYNAME          |B
IRTHDATE
-----|-----|-----|-----|-----|-----
27      |Cards     |Earl Hastings       |OF   |Speedy Earl          |1
982-04-22
73      |Giants    |Lester Johns        |P    |Big John              |1
984-06-09
99      |Giants    |Joe Bojangles        |C    |Little Joey           |1
991-07-11

3 rows selected
splice>
```

Running Splice Machine From a Shell Script

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

1. Create a shell script

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

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

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

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

SELECT * FROM Players;exit;EOF
```

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

2. Make your script executable

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

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

3. Run the script

In your terminal window, invoke the script:

```
./myShellScript.sh
```

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

Running Splice Machine Commands from a Shell Script...

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

Running Splice Machine SQL shell

```
For help: "splice> help;"splice> connect 'jdbc:splice://srv55:1527/splice
db;user=splice;password=admin';
```

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

0 rows inserted/updated/deleted

```
splice> INSERT INTO Players
```

```
VALUES (99, 'Giants', 'Joe Bojangles', 'C', 'Little Joey', '07/11/1991'),
```

```
(73, 'Giants', 'Lester Johns', 'P', 'Big John', '06/09/1984'),
```

```
(27, 'Cards', 'Earl Hastings', 'OF', 'Speedy Earl', '04/22/1982');
```

3 rows inserted/updated/deleted

```
splice> SELECT * FROM Players;
```

ID	TEAM	NAME	POS	DISPLAYNAME	BIRTHDATE
----	------	------	-----	-------------	-----------

27	Cards	Earl Hastings	OF	Speedy Earl	1982-04-22
73	Giants	Lester Johns	P	Big John	1984-06-09
99	Giants	Joe Bojangles	C	Little Joey	1991-07-11

3 rows selected

Using nohub for Long-Running Scripts

If you want to run an unattended shell script that may take a long time, you can: use the Unix `nohub` 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:

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

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

rlWrap Commands Synopsis

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

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

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

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

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

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

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

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

Uploading Your Data to an S3 Bucket

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

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

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

Create and Upload Data to an AWS S3 Bucket

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

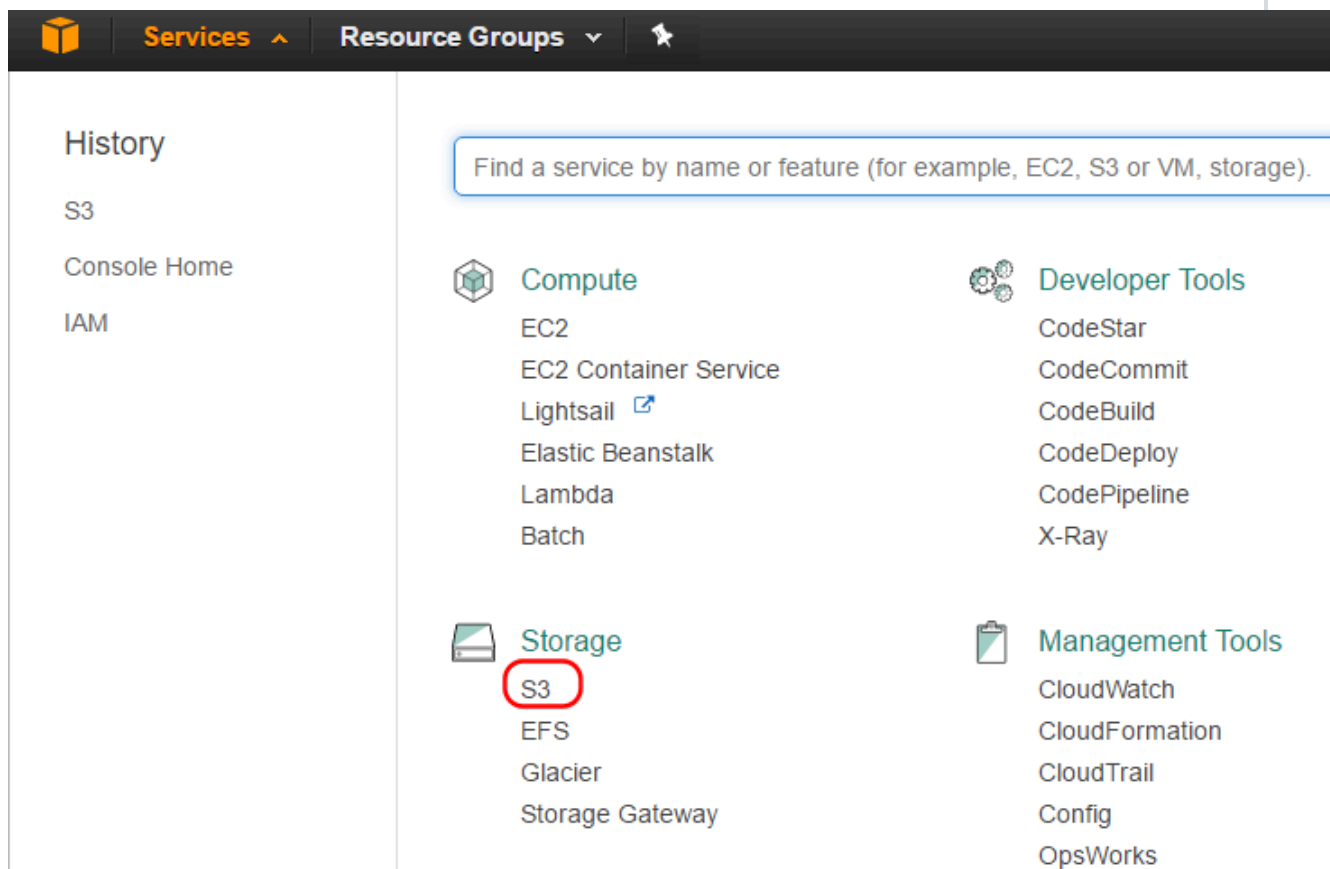
1. **Log in to the AWS Database Console**

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

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

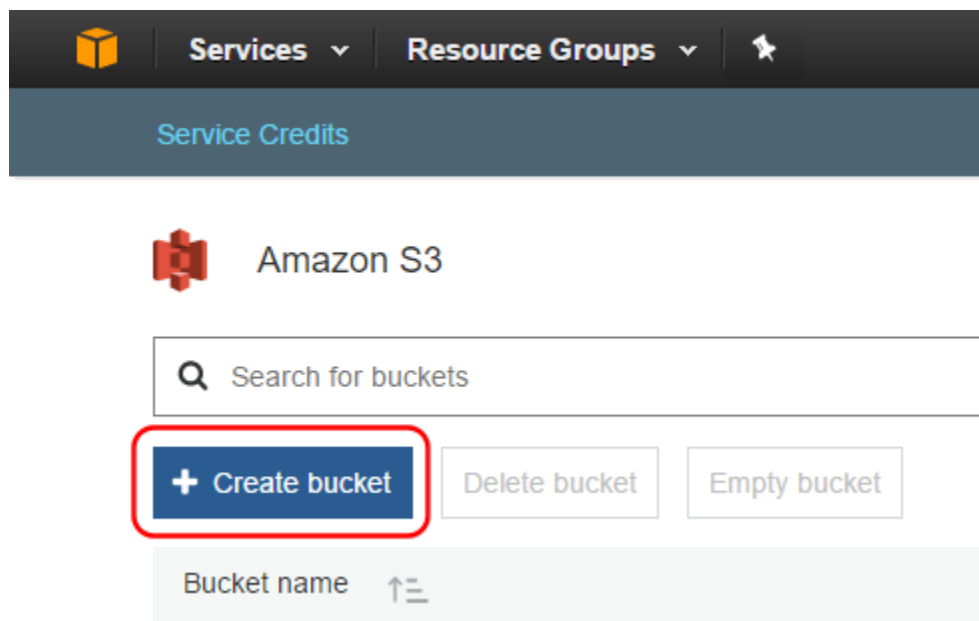


3. Select **S3** in the **Storage** section:



4. Create a new bucket

- 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:

Create bucket

1

Name and region

2

Set properties

3

Set permissions

4

Review

Name and region

Bucket name ⓘ

splicedocs-demos

Region

US West (N. California) ▼

Copy settings from an existing bucket

Select bucket (optional)

63 Buckets ▼

- c. Click the [Next](#) button to advance to the property settings for your new bucket:

Create bucket

1 ☒ Name and region 2 **Set properties** 3 ☐ Set permissions 4 ☐ Review

Versioning

Keep multiple versions of an object in the same bucket.

[Learn more](#)

☐ Disabled

Logging

Set up access log records that provide details about access requests.

[Learn more](#)

☐ Disabled

Tags

Use tags to track your cost against projects or other criteria.

[Learn more](#)

☐ 0 Tags

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:

Create bucket

✓ Name and region

✓ Set properties

3 Set permissions

4 Review

▼ Manage users

User ID	Objects	Object permissions
aws(Owner)	<input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Write	<input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Write

▼ Manage public permissions

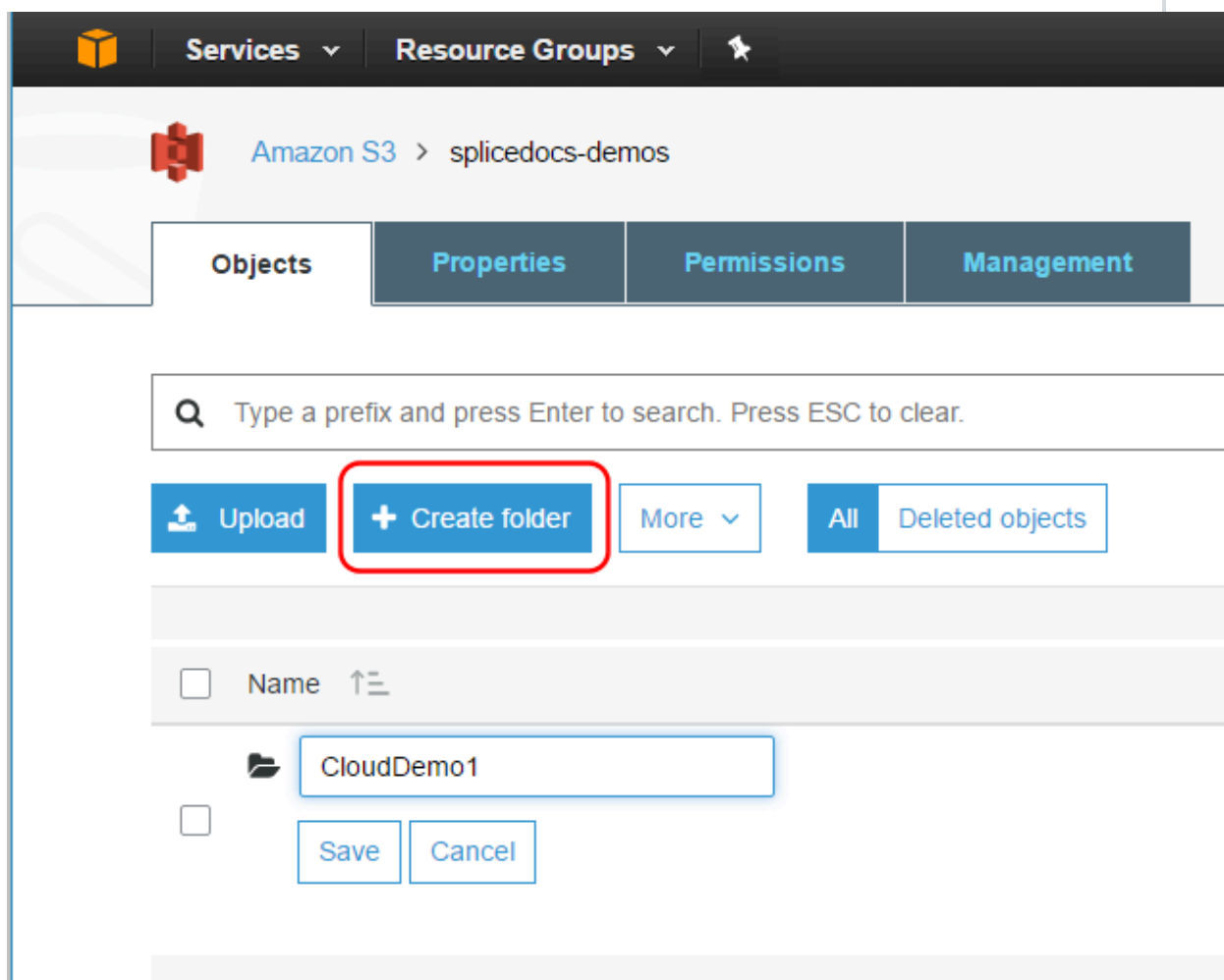
Group	Objects	Object permissions
Any authenticated AWS user	<input type="checkbox"/> Read <input type="checkbox"/> Write	<input type="checkbox"/> Read <input type="checkbox"/> Write
Everyone	<input type="checkbox"/> Read <input type="checkbox"/> Write	<input type="checkbox"/> Read <input type="checkbox"/> Write

- 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.

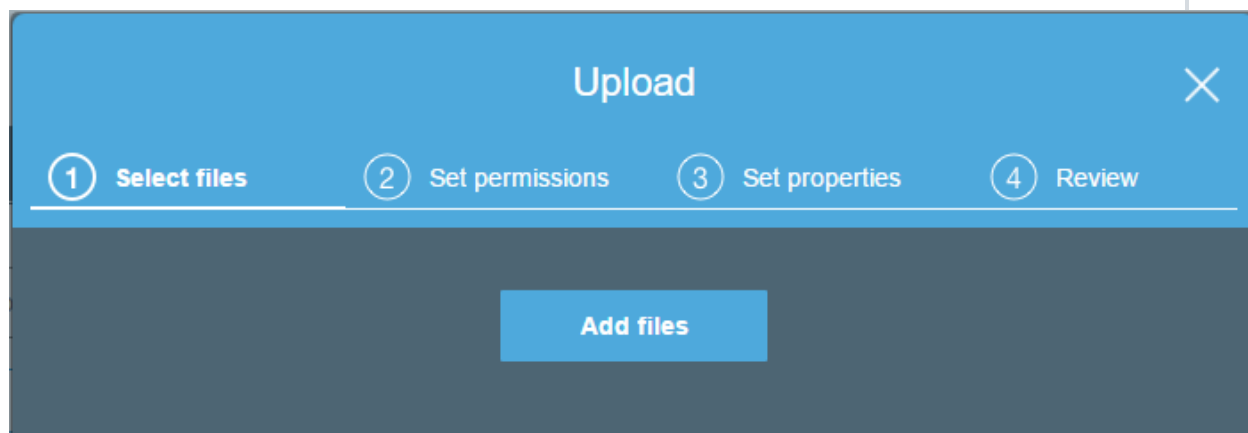
5. Upload data to your bucket

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.

Configuring an S3 Bucket for Splice Machine Access

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

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

Configure S3 Bucket Access

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

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

1. Log in to the AWS Database Console



You must have administrative access to configure S3 bucket access.

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




3. Access the IAM (Identify and Access Management) service:


Select **IAM** in the **Security, Identity & Compliance** section:


 **Services** ^ **Resource Groups** v 

History


Console Home

 **Compute**


- EC2
- EC2 Container Service
- Lightsail 
- Elastic Beanstalk
- Lambda
- Batch

 **Storage**


- S3
- EFS
- Glacier
- Storage Gateway


 **Database**

- RDS
- DynamoDB
- ElastiCache
- Redshift


 **Networking & Content Delivery**

- VPC
- CloudFront
- Direct Connect
- Route 53


 **Migration**

 **Developer Tools**

- CodeCommit
- CodeBuild
- CodeDeploy
- CodePipeline
- X-Ray

 **Management Tools**

- CloudWatch
- CloudFormation
- CloudTrail
- Config
- OpsWorks
- Service Catalog
- Trusted Advisor
- Managed Services

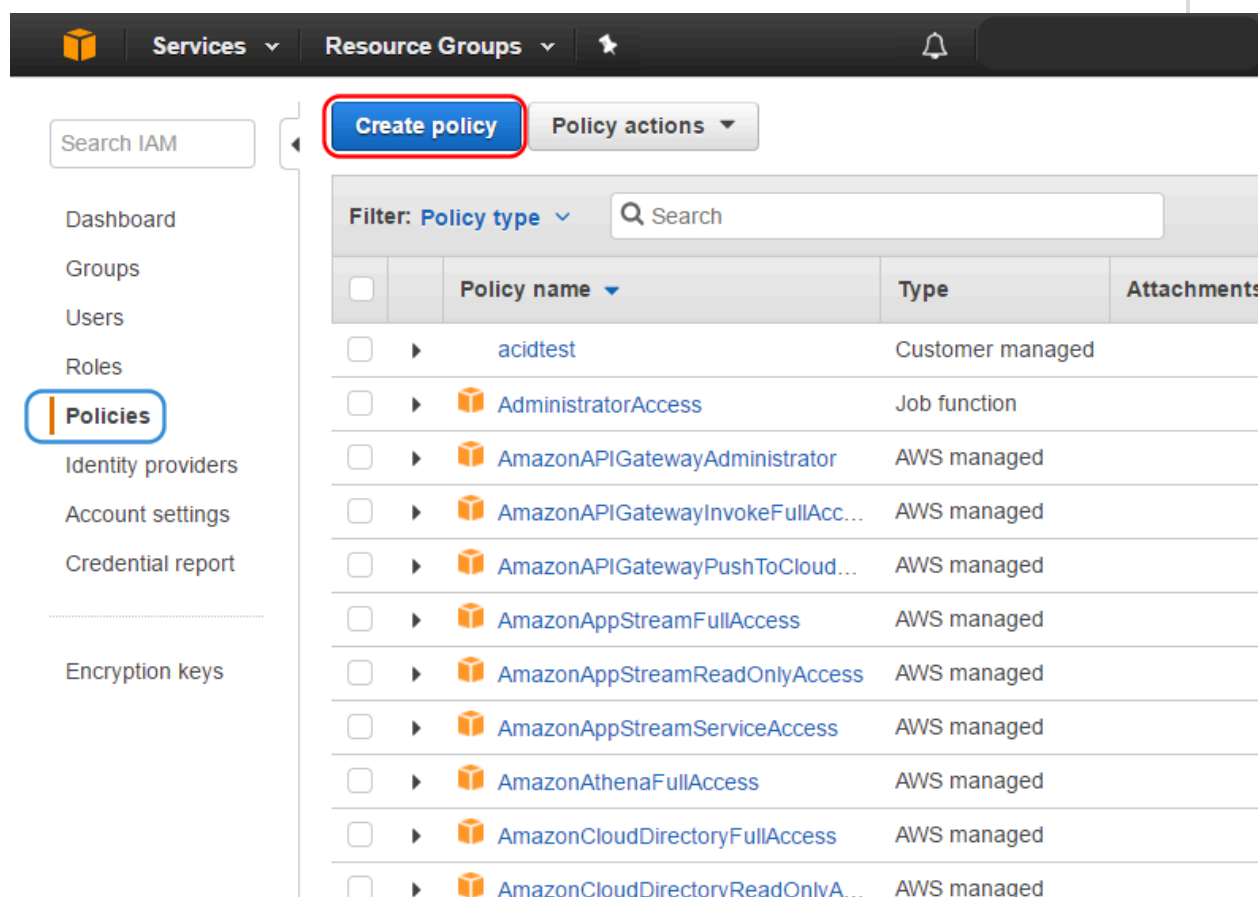
 **Security, Identity & Compliance**

- IAM**
- Inspector
- Certificate Manager
- Directory Service
- WAF & Shield
- Compliance Reports

-31-

4. Create a new policy:

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



The screenshot shows the AWS IAM console interface. At the top, there's a navigation bar with 'Services', 'Resource Groups', and a search icon. Below this, a left-hand sidebar contains a 'Search IAM' box and a list of navigation items: Dashboard, Groups, Users, Roles, **Policies** (highlighted with a blue box), Identity providers, Account settings, Credential report, and Encryption keys. The main content area is titled 'Policy actions' and features a 'Create policy' button, which is highlighted with a red rectangular box. Below the button, there's a table of existing policies. The table has columns for 'Filter: Policy type', a search box, 'Policy name', 'Type', and 'Attachments'. The table lists several AWS managed policies, including 'acidtest', 'AdministratorAccess', 'AmazonAPIGatewayAdministrator', 'AmazonAPIGatewayInvokeFullAcc...', 'AmazonAPIGatewayPushToCloud...', 'AmazonAppStreamFullAccess', 'AmazonAppStreamReadOnlyAccess', 'AmazonAppStreamServiceAccess', 'AmazonAthenaFullAccess', 'AmazonCloudDirectoryFullAccess', and 'AmazonCloudDirectoryReadOnlyA...'. Each row includes a checkbox and a right-pointing arrow.

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

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

Create Policy

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

Copy an AWS Managed Policy Select

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

Policy Generator Select

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

Create Your Own Policy Select

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

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

Review Policy

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

Policy Name

splice_access

Description

Allowing Splice Machine access to our S3 bucket.

Policy Document

1

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

```

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

```

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

Create Policy

Step 1 : [Create Policy](#)

Step 2 : [Set Permissions](#)

Step 3 : Review Policy

Review Policy

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

This policy is valid.

Policy Name
splice_access

Description
Allowing Splice Machine to access our S3 bucket

Policy Document

```

1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "s3:PutObject",
8         "s3:GetObject",
9         "s3:GetObjectVersion",
10        "s3:DeleteObject",

```

☒ Use autoformatting for policy editing

[Cancel](#) [Validate Policy](#) [Previous](#) [Create Policy](#)

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:

Add user

1

Details

2

Permissions

3

Review

4

Complete

Set user details

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

User name*

[Add another user](#)

Select AWS access type

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

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

* Required

[Cancel](#)

[Next: Permissions](#)

- d. Click **Attach existing policies directly**.
- e. Select the policy you just created and click **Next**:

Add user

1

Details

2

Permissions

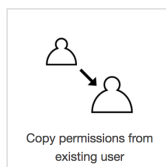
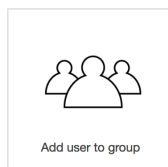
3

Review

4

Complete

Set permissions for SpliceMachine



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

[Create policy](#)

[Refresh](#)

Filter: Policy type

Search

Showing 250 results

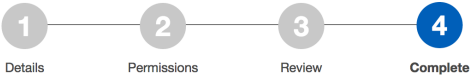
	Policy name	Type	Attachments	Description
<input type="checkbox"/>	SelfPasswordChange	Customer managed	1	
<input checked="" type="checkbox"/>	splice_access	Customer managed	0	


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


6. Save your access credentials

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

Add user



 **Success**
You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.
Users with AWS Management Console access can sign-in at: <https://sfc-samples.signin.aws.amazon.com/console>

 Download .csv

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

Close

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

Importing Data Into Your Splice Machine Database

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

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

Overview of Importing Data Into Your Database

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

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

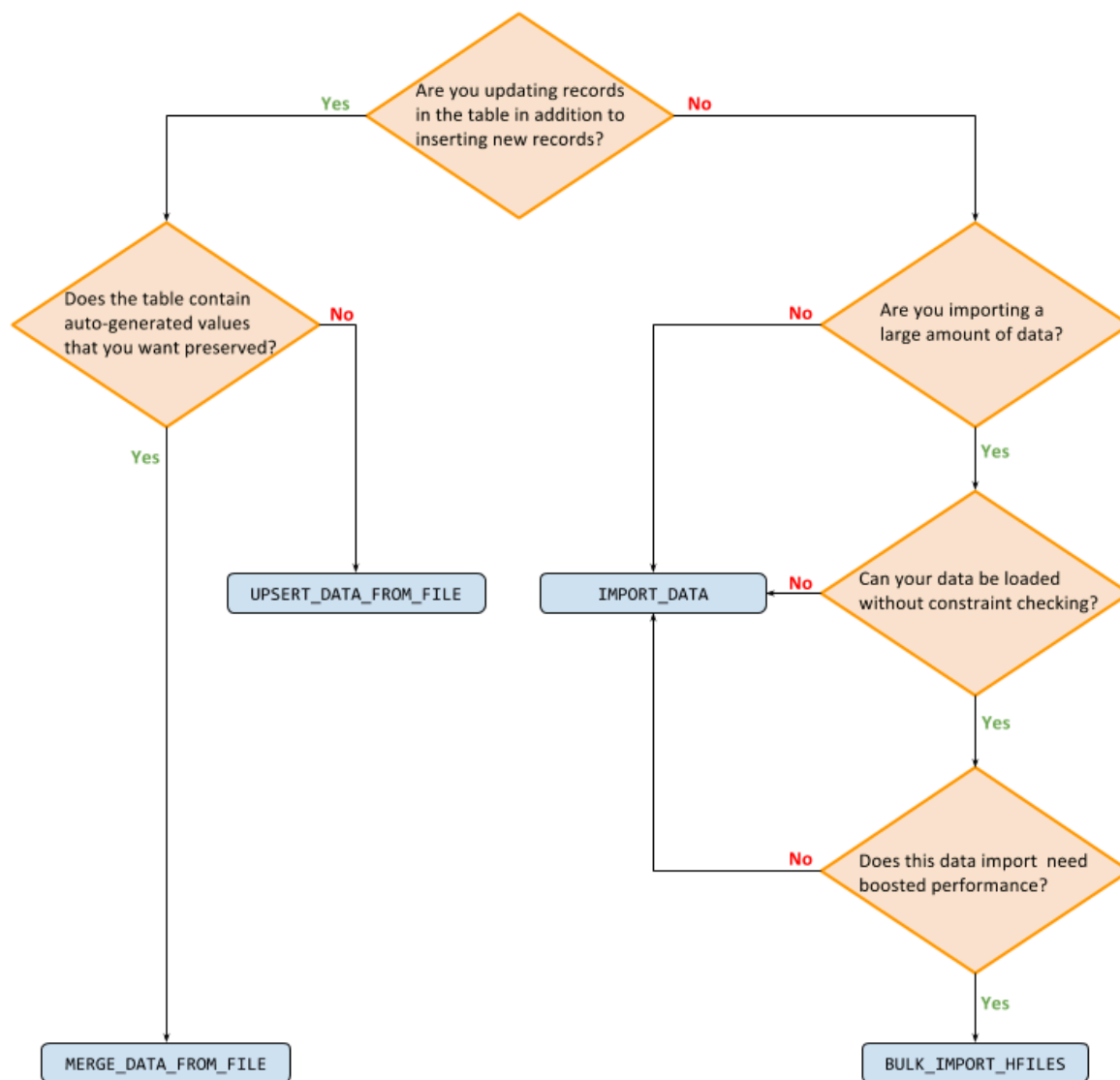
Data Import Procedures

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

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

Which Procedure Should I Use to Import My Data?

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



Notes

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

Import Procedures Syntax

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

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

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

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

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

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

And you'll find detailed reference descriptions of all four procedures in our [SQL Reference Manual](#).

See Also

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

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

Importing Data: Specifying the Import Parameter Values

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

Import Procedures Syntax

Three of our four data import procedures use identical parameters:

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

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


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

Overview of Parameters Used in Import Procedures

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

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

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

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

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

Import Parameters Reference

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

`schemaName`

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

Example: `SPLICE`

`tableName`

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

Example: `playerTeams`

`insertColumnList`

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

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

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

Example: `ID`, `TEAM`

See [Importing and Updating Records](#) for additional information about handling of missing, generated, and default values during data importation.

fileOrDirectoryName

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

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

NOTE:

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

Example: `data/mydata/mytable.csv`

Importing from S3

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

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

See [Specifying Your Input Data Location](#) for additional information about specifying your input data location.

Importing Compressed Files

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

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

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

oneLineRecords

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

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

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

Example: `true`

charset

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

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

Example: `null`

columnDelimiter

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

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

Special character	Display
<code>\t</code>	Tab

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

Example: ' | '

See [Column Delimiters](#) for additional information about column delimiters.

characterDelimiter

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

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

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

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

Notes:

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

Example:

See [Character Delimiters](#) for additional information about character delimiters.

timestampFormat

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

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



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

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

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

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

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

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

Timestamps and Importing Data at Different Locations

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

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

Examples

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

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

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

See [Time and Date Formats in Input Records](#) for additional information about date, time, and timestamp values.

dateFormat

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

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

Example: `yyyy-MM-dd`

See [Time and Date Formats in Input Records](#) for additional information about date, time, and timestamp values.

timeFormat

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

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

Example: `HH:mm:ss`

See [Time and Date Formats in Input Records](#) for additional information about date, time, and timestamp values.

badRecordsAllowed

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

These values have special meaning:

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

Example: `25`

badRecordDirectory

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

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

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

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

Example: `'importErrsDir'`

bulkImportDirectory

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

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

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

Please review the [Bulk HFile Import Walkthrough](#) topic to understand how importing bulk HFiles works.

skipSampling

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

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

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

Example: `false`

Please review the [Bulk HFile Import Walkthrough](#) topic to understand how importing bulk HFiles works.

See Also

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

Importing Data: Input Considerations

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

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

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

This topic includes the following sections:

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

Specifying Your Input Data Location

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

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

Standalone Version Input File Path

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

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

HBase Input File Path

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

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

AWS S3 Input File Path

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

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

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

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

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

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

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

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

Input Data File Format

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

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

Importing Compressed Files

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

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

Importing Multi-line Records

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

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

Importing Large Datasets in Groups of Files

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

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

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

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

Delimiters in Your Input Data

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

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

Use Special Characters for Delimiters

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

Column Delimiters

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

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

Special character	Display
\t	Tab
\f	Formfeed
\b	Backspace
\\	Backslash
^a (or ^A)	Control-a <div data-bbox="399 899 1312 1032"> <p>NOTE: If you are using a script file from the <code>splice></code> command line, your script can contain the actual <code>Control-a</code> character as the value of this parameter.</p> </div>

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

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

Character Delimiters

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

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

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

Time and Date Formats in Input Records

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

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

[<https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html>][1]{: target="_blank"}

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

All Values Must Use the Same Format

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

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

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

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

Additional Notes

A few additional notes:

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

» Detailed information about each of these data types is found in our SQL Reference Manual:

» [Timestamp Data Type](#)

» [Date Data Type](#)

» [Time Data Type](#)

Importing and Updating Records

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

» [Inserting and Updating Column Values When Importing Data](#)

» [Inserting and Updated Generated or Default Values](#)

» [Handling Missing Values](#)

Inserting and Updating Column Values When Importing Data

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

» Whether you are importing a new record or updating an existing record.

» If the column is specified in your `insertColumnList` parameter.

» If the table column is a generated value or has a default value.

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

» For newly inserted records, the default or auto-generated value is always inserted, as usual.

» If you are updating an existing record in the table with UPSERT, the default auto-generated value in that record is overwritten with a new value.

» If you are updating an existing record in the table with MERGE, the column value is not updated.

Importing a New Record Into a Database Table

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

Column included in <code>importColumnList</code> ?	Table column conditions	Action Taken
YES	N/A	Import value inserted into table column if valid; if not valid, a bad record error is logged.
NO	Has Default Value	Default value is inserted into table column.

Column included in importColumnList?	Table column conditions	Action Taken
	Is Generated Value	Generated value is inserted into table column.
	None	NULL is inserted into table column.

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

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

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

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

» If the target column in your database has a string data type, such as CHAR or VARCHAR, the literal value "DEFAULT" is

inserted into your database..

- » If the target column is not a string data type, an error will occur.

Importing Into a Table that Contains Generated or Default Values

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

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

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

Updating a Table Record with UPSERT

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

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

Updating a Table Record with MERGE

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

Column included in <i>importColumnList</i> ?	Table column conditions	Action Taken
YES	N/A	Import value updated in table column if valid; if not valid, a bad record error is logged.
NO	N/A	Table column is not updated.

Importing CLOBs and BLOBs

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

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

Scripting Your Imports

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

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

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

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

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

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

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

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

See Also

- » [Importing Data: Tutorial Overview](#)
- » [Importing Data: Input Parameters](#)
- » [Importing Data: Error Handling](#)
- » [Importing Data: Usage Examples](#)
- » [Importing Data: Bulk HFile Examples](#)
- » [Importing Data: Importing TPCD Data](#)
- » [SYSCS_UTIL.IMPORT_DATA](#)
- » [SYSCS_UTIL.INSERT_DATA_FROM_FILE](#)

» [SYSCS_UTIL.MERGE_DATA_FROM_FILE](#)

» [SYSCS_UTIL.BULK_IMPORT_HFILE](#)

Importing Data: Logging and Error Handling

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

Logging

Each of these import procedures includes a logging facility:

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

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

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

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

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

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

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

Stopping the Import Due to Too Many Errors

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

These `badRecordsAllowed` values have special meaning:

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

Managing Logging When Importing Multiple Files

In addition to importing a single file, the [SYSCS_UTIL.IMPORT_DATA](#) and [SYSCS_UTIL.UPSERT_DATA_FROM_FILE](#) procedures can import all of the files in a directory.

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

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

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

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

See Also

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

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

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

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

Import, Upsert, or Merge?

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

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

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

A fourth option works differently:

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

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

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

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

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

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

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

And your data file looks like this:

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

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

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

How Missing Columns are Handled With an Insert Column List

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

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

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

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

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

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

```
INSERT INTO testUpsert(a1,b1) VALUES (1,1), (2,2), (3,3), (6,6);
```

```
splice> select * from testUpsert;
```

A1	B1	C1	D1
1	1	1	999
2	2	2	999
3	3	3	999
6	6	4	999

```
4 rows selected
```

```
INSERT INTO testMerge (a1,b1) VALUES (1,1), (2,2), (3,3), (6,6);
```

```
splice> select * from testMerge;
```

A1	B1	C1	D1
1	1	1	999
2	2	2	999
3	3	3	999
6	6	4	999

```
4 rows selected
```

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

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

```
0|0
1|2
2|4
3|6
4|8
```

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

```
CALL SYSCS_UTIL.UPsert_DATA_FROM_FILE('TEST','testUpsert','a1,b1','/Users/garyh/Documents/ttest.csv','|',null,null,null,null,0,'/var/tmp/bad/',false,null);
```

rowsImported	failedRows	files	dataSize	failedLog
5	0	1	20	NONE

```
splice> SELECT * FROM testUpsert;
```

A1	B1	C1	D1
0	0	10001	999
1	2	10002	999
2	4	10003	999
3	6	10004	999
4	8	10005	999
6	6	4	999

6 rows selected

```
CALL SYSCS_UTIL.MERGE_DATA_FROM_FILE('TEST','testMerge','a1,b1','/Users/garyh/Documents/ttest.csv','|',null,null,null,null,0,'/var/tmp/bad/',false,null);
```

rowsUpdated	rowsInserted	failedRows	files	dataSize	failedLog
3	2	0	1	2	NONE

```
splice> select * from testMerge;
```

A1	B1	C1	D1
0	0	10001	999
1	2	1	999
2	4	2	999
3	6	3	999
4	8	10002	999
6	6	4	999

6 rows selected

You'll notice that:

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

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

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

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

Example 4: Specifying a timestamp format for an entire table

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

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

You can then import the data with the following call:

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



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

Example 5: Importing strings with embedded special characters

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

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

We then use the following call to import the data:

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

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

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

Example 6: Using single quotes to delimit strings

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

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

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

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

See Also

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

Importing Data With the Bulk HFile Import Procedure

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

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

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



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

How Importing Your Data as HFiles Works

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

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

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

You have two choices for determining the table splits:

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

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

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

Configuration Settings

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

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

Importing Data From the Cloud

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

Manually Computing Table Splits

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

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

-or-

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

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

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

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

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

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

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

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

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

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

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

Examples of Using SYSCS_UTIL.BULK_IMPORT_HFILE

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

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

Example 1: Bulk HFile Import with Automatic Table Splitting

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

Follow these steps:

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

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

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

2. Create table and index:

```
CREATE TABLE TPCH.LINEITEM (
  L_ORDERKEY BIGINT NOT NULL,
  L_PARTKEY INTEGER NOT NULL,
  L_SUPPKEY INTEGER NOT NULL,
  L_LINENUMBER INTEGER NOT NULL,
  L_QUANTITY DECIMAL(15,2),
  L_EXTENDEDPRICE DECIMAL(15,2),
  L_DISCOUNT DECIMAL(15,2),
  L_TAX DECIMAL(15,2),
  L_RETURNFLAG VARCHAR(1),
  L_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. Import the HFiles Into Your Database

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

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

The generated HFiles are automatically deleted after being imported.

Example 2: Using `SPLIT_TABLE_OR_INDEX` to Compute Table Splits

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

Follow these steps:

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

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

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

2. Create table and index:

```

CREATE TABLE TPCH.LINEITEM (
  L_ORDERKEY BIGINT NOT NULL,
  L_PARTKEY INTEGER NOT NULL,
  L_SUPPKEY INTEGER NOT NULL,
  L_LINENUMBER INTEGER NOT NULL,
  L_QUANTITY DECIMAL(15,2),
  L_EXTENDEDPRICE DECIMAL(15,2),
  L_DISCOUNT DECIMAL(15,2),
  L_TAX DECIMAL(15,2),
  L_RETURNFLAG VARCHAR(1),
  L_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. Compute the split row keys for your table and set up the split in your database:

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

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

```

1500000 |
3000000 |
4500000 |

```

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

```

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

```

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

```
'L_ORDERKEY,L_LINENUMBER'
```

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

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

4. Compute the split keys for your index:

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

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

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

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

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

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

5. Import the HFiles Into Your Database

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

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

The generated HFiles are automatically deleted after being imported.

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

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

Follow these steps:

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

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

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

2. Create table and index:

```
CREATE TABLE TPCH.LINEITEM (
    L_ORDERKEY BIGINT NOT NULL,
    L_PARTKEY INTEGER NOT NULL,
    L_SUPPKEY INTEGER NOT NULL,
    L_LINENUMBER INTEGER NOT NULL,
    L_QUANTITY DECIMAL(15,2),
    L_EXTENDEDPRICE DECIMAL(15,2),
    L_DISCOUNT DECIMAL(15,2),
    L_TAX DECIMAL(15,2),
    L_RETURNFLAG VARCHAR(1),
    L_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)
);
```

3. Compute the split row keys for the table:

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

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

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

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

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

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

```
'L_ORDERKEY,L_LINENUMBER'
```

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

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

4. Set up the table splits in your database:

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

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

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

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

5. Compute the split keys for your index:

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

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

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

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

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

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

6. Set up the indexes in your database:

- a. Copy the row key values from the output file:

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

- b. Now call our system procedure to split the index:

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

7. Import the HFiles Into Your Database

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


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

The generated HFiles are automatically deleted after being imported.

See Also

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

Importing TPCB Data Into Your Database

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

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

Import TPCB Data

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

- 1. Create the schema and tables**

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

```

CREATE SCHEMA TPCH;

CREATE TABLE TPCH.LINEITEM (
  L_ORDERKEY BIGINT NOT NULL,
  L_PARTKEY INTEGER NOT NULL,
  L_SUPPKEY INTEGER NOT NULL,
  L_LINENUMBER INTEGER NOT NULL,
  L_QUANTITY DECIMAL(15,2),
  L_EXTENDEDPRICE DECIMAL(15,2),
  L_DISCOUNT DECIMAL(15,2),
  L_TAX DECIMAL(15,2),
  L_RETURNFLAG VARCHAR(1),
  L_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 TPCB data in an AWS S3 bucket for convenient retrieval. You can copy/paste the following `SYSCS_UTIL.IMPORT_DATA` statements to quickly pull that data into your database:

```

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

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

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

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

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

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

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

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

```

3. Run a query

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

```
-- QUERY 01
select
  l_returnflag,
  l_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('1998-12-0
1 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.

Importing Your Own Data

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

The TPCH Queries

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

Query01

```
-- QUERY 01
select
  l_returnflag,
  l_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
```

Query02

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


Query03

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

Query04

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

Query05

```
-- QUERY 05
select
    n_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue
from
    TPCH.customer,
    TPCH.orders,
    TPCH.lineitem,
    TPCH.supplier,
    TPCH.nation,
    TPCH.region
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and l_suppkey = s_suppkey
    and c_nationkey = s_nationkey
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'ASIA'
    and o_orderdate >= date('1994-01-01')
    and o_orderdate < date({fn TIMESTAMPADD(SQL_TSI_YEAR, 1, cast('19
94-01-01 00:00:00' as timestamp))})
group by
    n_name
order by
    revenue desc
-- END OF QUERY
```

Query06

```
-- QUERY 06
select
    sum(l_extendedprice * l_discount) as revenue
from
    TPCH.lineitem
where
    l_shipdate >= date('1994-01-01')
    and l_shipdate < date({fn TIMESTAMPADD(SQL_TSI_YEAR, 1, cast('199
4-01-01 00:00:00' as timestamp))})
    and l_discount between .06 - 0.01 and .06 + 0.01
    and l_quantity > 24
-- END OF QUERY
```

Query07

```

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

```

Query08

```

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

```

Query09

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

Query10

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

Query11

```
-- QUERY 11
select
    ps_partkey,
    sum(ps_supplycost * ps_availqty) as value
from
    TPCH.partsupp,
    TPCH.supplier,
    TPCH.nation
where
    ps_suppkey = s_suppkey
    and s_nationkey = n_nationkey
    and n_name = 'GERMANY'
group by
    ps_partkey having
        sum(ps_supplycost * ps_availqty) > (
            select
                sum(ps_supplycost * ps_availqty) * 0.0001000000
            from
                TPCH.partsupp,
                TPCH.supplier,
                TPCH.nation
            where
                ps_suppkey = s_suppkey
                and s_nationkey = n_nationkey
                and n_name = 'GERMANY'
        )
order by
    value desc
-- END OF QUERY
```


Query12

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

Query13

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

Query14

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

Query15

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

Query16

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

Query17

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

Query18

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

Query19

```

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

```

Query21

```

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

```


Query22

```

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

```

See Also

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

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

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.

Integrating Apache Storm with Splice Machine

This topic walks you through building and running two different examples of integrating Storm with Splice Machine:

- » [Inserting Random Values in Splice Machine with Storm](#)
- » [Inserting Data into Splice Machine from MySQL](#)

Inserting Random Values in Splice Machine with Storm

This example iterates through a list of random words and numbers, inserting the values into a Splice Machine database; follow along in these steps:

1. Download the Sample Code:

Pull the code from our git repository:

```
https://github.com/splicemachine/splice-community-sample-code/tree/master/tutorial-storm
```

2. Check the prerequisites:

You must be running Splice Machine 2.0 or later on the same machine on which you will run this example code. If Splice Machine is running on a different machine, you'll need to modify the `server` variable in the `SpliceDumperTopology.java` file; change it to the name of the server that is running Splice Machine.

You also must have `maven v.3.3.9` or later installed.

3. Create the test table in Splice Machine:

This example inserts data into a Splice Machine table named `testTable`. You need to create that table by entering this command at the `splice>` prompt:

```
splice> CREATE TABLE testTable( word VARCHAR(100), number INT );
```

4. Compile the sample code:

Compile the sample code with this command

```
% mvn clean compile dependency:copy-dependencies
```

5. Run the sample code:

Follow these steps:

- a. Make sure that Splice Machine is running and that you have created the `testTable` table.
- b. Execute this script to run the program:

```
% run-storm.sh
```

- c. Query `testTable` in Splice Machine to verify that it has been populated with random words and numbers:

```
splice> select * from testTable;
```

About the Sample Code Classes

The random insertion example contains the following java classes, each of which is described below:

Class	Description
<code>SpliceCommunicator.java</code>	Contains methods for communicating with Splice Machine.
<code>SpliceConnector.java</code>	Establishes a JDBC connection with Splice Machine.
<code>SpliceDumperBolt.java</code>	Dumps data into Splice Machine.
<code>SpliceDumperTopology.java</code>	Defines the Storm topology for this example.
<code>SpliceIntegerSpout.java</code>	Emits tuples that are inserted into the Splice Machine table.

Inserting Data into Splice Machine from MySQL

This example uses Storm to read data from a MySQL database, and insert that data into a table in Splice Machine.

1. Download the Sample Code:

Pull the code from our git repository:

```
https://github.com/splicemachine/splice-community-sample-code/tree/master/tutorial-storm
```

2. Check the prerequisites:

You must be running Splice Machine 2.0 or later on the same machine on which you will run this example code. If Splice Machine is running on a different machine, you'll need to modify the `server` variable in the `MySQLToSpliceTopology.java` file; change it to the name of the server that is running Splice Machine.

You also must have `maven v.3.3.9` or later installed.

This example assumes that your MySQL database instance is running on the same machine on which you're running Splice Machine, and that the root user does not have a password. If either of these is not true, then you need to modify the call to the `seedBufferQueue` method in the `MySQLSpout.java` file. This method takes four parameters that you may need to change:

```
seedBufferQueue( MySQLServer, MySQLDatabase, mySqlUserName, mySqlPassword );
```

The default settings used in this example are:

```
seedBufferQueue( "localhost", "test", "root", "" );
```

3. Create the students table in Splice Machine:

This example inserts data into a Splice Machine table named `students`. You need to create that table by entering this command at the `splice>` prompt:

```
splice> CREATE TABLE students( name VARCHAR(100) );
```

4. Create the students table in your MySQL database:

This example read data from a MySQL table named `students`. You need to create that table in MySQL:

```
$$ CREATE TABLE students( id INTEGER, name VARCHAR(100) );
```

If your MySQL instance is on a different machine

5. Compile the sample code:

Compile the sample code with this command

```
% mvn clean compile dependency:copy-dependencies
```

6. Run the sample code:

Follow these steps:

a. Make sure that Splice Machine is running and that you have created the `testTable` table.

b. Execute this script to run the program:

```
% run-mysql-storm.sh
```

c. Query the `students` table in Splice Machine to verify that it has been populated with data from the MySQL table:

```
splice> select * from students;
```

About the Sample Code Classes

This example contains the following java classes:

Class	Description
<code>MySQLCommunicator.java</code>	Contains methods for communicating with MySQL.
<code>MySQLConnector.java</code>	Establishes a JDBC connection with MySQL.
<code>MySQLSpliceBolt.java</code>	Dumps data from MySQL into Splice Machine.
<code>MySQLSpout.java</code>	Emits tuples from MySQL that are inserted into the Splice Machine table.
<code>MySQLToSpliceTopology.java</code>	Defines the Storm topology for this example.
<code>SpliceCommunicator.java</code>	Contains methods for communicating with Splice Machine.
<code>SpliceConnector.java</code>	Establishes a JDBC connection with Splice Machine.

Streaming MQTT Spark Data

This topic walks you through using MQTT Spark streaming with Splice Machine. MQTT is a lightweight, publish-subscribe messaging protocol designed for connecting remotely when a small footprint is required. MQTT is frequently used for data collection with the Internet of Things (IoT).

The example code in this tutorial uses [Mosquitto](#), which is an open source message broker that implements the MQTT. This tutorial uses a cluster managed by MapR; if you're using different platform management software, you'll need to make a few adjustments in how the code is deployed on your cluster.

NOTE: All of the code used in this tutorial is available in our [GitHub community repository](#).

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

Watch the Video

The following video shows you how to:

- » put messages on an MQTT queue
- » consume those messages using Spark streaming
- » save those messages to Splice Machine with a virtual table (VTI)

Written Walk Through

This section walks you through the same sequence of steps as the video, in these sections:

- » [Deploying the Tutorial Code](#) walks you through downloading and deploying the sample code.
- » [About the Sample Code](#) describes the high-level methods in each class.
- » [About the Sample Code Scripts](#) describes the scripts used to deploy and execute the sample code.

Deploying the Tutorial Code

Follow these steps to deploy the tutorial code:

1. **Download the code from our [GitHub community repository](#).**
Pull the code from our git repository:

```
https://github.com/splicemachine/splice-community-sample-code/tree/master/tutorial-mqtt-spark-streaming
```

2. Compile and package the code:

```
mvn clean compile package
```

3. Copy three JAR files to each server:

Copy these three files:

```
./target/splice-tutorial-mqtt-2.0.jar  
spark-streaming-mqtt_2.10-1.6.1.jar  
org.eclipse.paho.client.mqttv3-1.1.0.jar
```

to this directory on each server:

```
/opt/splice/default/lib
```

4. Restart Hbase

5. Create the target table in splice machine:

Run this script to create the table:

```
create-tables.sql
```

6. Start Mosquitto:

```
sudo su /usr/sbin/mosquitto -d -c /etc/mosquitto/mosquitto.conf > /var/log/mosquitto.log 2>&1
```

7. Start the Spark streaming script:

```
sudo -su mapr ./run-mqtt-spark-streaming.sh tcp://srv61:1883 /testing 10
```

The first parameter (`tcp://srv61:1883`) is the MQTT broker, the second (`/testing`) is the topic name, and the third (`10`) is the number of seconds each stream should run.

8. Start putting messages on the queue:

Here's a java program that is set up to put messages on the queue:

```
java -cp /opt/splice/default/lib/splice-tutorial-mqtt-2.0-SNAPSHOT.jar:/opt/splice/default/lib/org.eclipse.paho.client.mqttv3-1.1.0.jar com.splicemachine.tutorials.sparkstreaming.mqtt.MQTTPublisher tcp://localhost:1883 /testing 1000 R1
```

The first parameter (`tcp://localhost:1883`) is the MQTT broker, the second (`/testing`) is the topic name, the third (`1000`) is the number of iterations to execute, and the fourth parameter (`R1`) is a prefix for this run.

NOTE: The source code for this utility program is in a different GitHub project than the rest of this code. You'll find it in the [tutorial-kafka-producer](#) Github project.

About the Sample Code

This section describes the main class methods used in this MQTT example code; here's a summary of the classes:

Java Class	Description
MQTTPublisher	Puts csv messages on an MQTT queue.
SparkStreamingMQTT	The Spark streaming job that reads messages from the MQTT queue.
SaveRDD	Inserts the data into Splice Machine using the <code>RFIDMessageVTI</code> class.
RFIDMessageVTI	A virtual table interface for parsing an <code>RFIDMessage</code> .
RFIDMessage	Java object (a POJO) for converting from a csv string to an object to a database entry.

MQTTPublisher

This class puts CSV messages on an MQTT queue. The function of most interest in `MQTTPublisher.java` is `DoDemo`, which controls our sample program:

```

public void doDemo() {
    try {
        long startTime = System.currentTimeMillis();
        client = new MqttClient(broker, clientId);
        client.connect();
        MqttMessage message = new MqttMessage();
        for (int i=0; inumMessages; i++) {
            // Build a csv string
            message.setPayload( prefix + "Asset" + i + ", Location" + i + "," + new Timestamp((new Date()).getTime()).getBytes());
            client.publish(topicName, message);
            if (i % 1000 == 0) {
                System.out.println("records:" + i + " duration=" + (System.currentTimeMillis() - startTime));
                startTime = System.currentTimeMillis();
            }
            client.disconnect();
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }
}

```

DoDemo does a little initialization, then starts putting messages out on the queue. Our sample program is set up to loop until it creates numMessages messages; after every 1000 messages, it displays a status message that helps us determine how much time is going to put messages on the queue, and how much to take them off the queue.

DoDemo builds a csv record (line) for each message, setting an asset ID, a location ID, and a timestamp in the payload of the message. It then publishes that message to the topic topicName.

SparkStreamingMQTT

Once the messages are on the queue, our SparkStreamingMQTT class object reads them from the queue and inserts them into our database. The main method in this class is processMQTT:

```

public void processMQTT(final String broker, final String topic, final int numSeconds) {

    LOG.info("***** SparkStreamingMQTTOutside.processMQTT start");

    // Create the spark application and set the name to MQTT
    SparkConf sparkConf = new SparkConf().setAppName("MQTT");

    // Create the spark streaming context with a 'numSeconds' second batch size
    jssc = new JavaStreamingContext(sparkConf, Durations.seconds(numSeconds));
    jssc.checkpoint(checkpointDirectory);

    LOG.info("***** SparkStreamingMQTTOutside.processMQTT about to read the MQTTUtils.createStream");
    //2. MQTTUtils to collect MQTT messages
    JavaReceiverInputDStreamString> messages = MQTTUtils.createStream(jssc, broker,
topic);

    LOG.info("***** SparkStreamingMQTTOutside.processMQTT about to do foreachRDD");
    //process the messages on the queue and save them to the database
    messages.foreachRDD(new SaveRDD());

    LOG.info("***** SparkStreamingMQTTOutside.processMQTT prior to context.start");
    // Start the context
    jssc.start();
    jssc.awaitTermination();
}

```

The `processMQTT` method takes three parameters:

broker

The URL of the MQTT broker.

topic

The MQTT topic name.

numSeconds

The number of seconds at which streaming data will be divided into batches.

The `processMQTT` method processes the messages on the queue and saves them by calling the `SaveMDD` class.

SaveRDD

The `SaveRDD` class is an example of a Spark streaming function that uses our virtual table interface (VTI) to insert data into your Splice Machine database. This function checks for messages in the stream, and if there any, it creates a connection your database and uses a prepared statement to insert the messages into the database.

```

/**
 * This is an example of spark streaming function that
 * inserts data into Splice Machine using a VTI.
 *
 * @author Erin Driggers
 */

public class SaveRDD implements FunctionJavaRDDString>, Void>, Externalizable {

    private static final Logger LOG = Logger.getLogger(SaveRDD.class);

    @Override
    public Void call(JavaRDDString> rddRFIDMessages) throws Exception {
        LOG.debug("About to read results:");
        if (rddRFIDMessages != null & rddRFIDMessages.count() > 0) {
            LOG.debug("Data to process:");
            //Convert to list
            ListString> rfidMessages = rddRFIDMessages.collect();
            int numRclds = rfidMessages.size();

            if (numRclds > 0) {
                try {
                    Connection con = DriverManager.getConnection("jdbc:splice://localhost:1527/splicedb;user=splice;password=admin");

                    //Syntax for using a class instance in a VTI, this could also be a table function
                    String vtiStatement = "INSERT INTO IOT.RFID "
                        + "select s.* from new com.splicemachine.tutorials.sparkstreaming.mqtt.RFIDMessageVTI(?) s ("
                        + RFIDMessage.getTableDefinition() + ")";
                    PreparedStatement ps = con.prepareStatement(vtiStatement);
                    ps.setObject(1, rfidMessages);
                    ps.execute();
                } catch (Exception e) {
                    //It is important to catch the exceptions as log messages because it is difficult
                    //to trace what is happening otherwise
                    LOG.error("Exception saving MQTT records to the database" + e.getMessage(), e);
                } finally {
                    LOG.info("Complete insert into IOT.RFID");
                }
            }
        }
        return null;
    }
}

```

The heart of this function is the statement that creates the prepared statement, using a VTI class instance:

```
String vtiStatement = "INSERT INTO IOT.RFID " + "select s.* from new com.splicemachine.tutorials.sparkstreaming.mqtt.RFIDMessageVTI(?) s ("
    + RFIDMessage.getTableDefinition() + ")";
PreparedStatement ps = con.prepareStatement(vtiStatement);
```

Note that the statement references both our `RFIDMessage` and `RFIDMessageVTI` classes, which are described below.

RFIDMessageVTI

The `RFIDMessageVTI` class implements an example of a virtual table interface that reads in a list of strings that are in CSV format, converts that into an `RFIDMessage` object, and returns the resultant list in a format that is compatible with Splice Machine.

This class features an override of the `getDataSet` method, which loops through each CSV record from the input stream and converts it into an `RFIDMessage` object that is added onto a list of message items:

```
@Override
public DataSetLocatedRow> getDataSet(SpliceOperation op, DataSetProcessor dsp, ExecRow execRow) throws StandardException {
    operationContext = dsp.createOperationContext(op);

    //Create an arraylist to store the key / value pairs
    ArrayListLocatedRow> items = new ArrayListLocatedRow>();

    try {

        int numRcds = this.records == null ? 0 : this.records.size();

        if (numRcds > 0) {

            LOG.info("Records to process:" + numRcds);
            //Loop through each record convert to a SensorObject
            //and then set the values
            for (String csvString : records) {
                CsvBeanReader beanReader = new CsvBeanReader(new StringReader(csvString), CsvPreference.STANDARD_PREFERENCE);
                RFIDMessage msg = beanReader.read(RFIDMessage.class, header, processors);

                items.add(new LocatedRow(msg.getRow()));
            }
        }
    } catch (Exception e) {
        LOG.error("Exception processing RFIDMessageVTI", e);
    } finally {
        operationContext.popScope();
    }
    return new ControlDataSet>(items);
}
```



For more information about using our virtual table interface, see [Using the Splice Machine Virtual Table Interface](#).

RFIDMessage

The `RFIDMessage` class creates a simple Java object (a POJO) that represents an RFID message; we use this to convert an incoming CSV-formatted message into an object. This class includes getters and setters for each of the object properties, plus the `getTableDefinition` and `getRow` methods:

```
/**
 * Used by the VTI to build a Splice Machine compatible resultset
 *
 * @return
 * @throws SQLException
 * @throws StandardException
 */
public ValueRow getRow() throws SQLException, StandardException {
    ValueRow valueRow = new ValueRow(5);
    valueRow.setColumn(1, new SQLVarchar(this.getAssetNumber()));
    valueRow.setColumn(2, new SQLVarchar(this.getAssetDescription()));
    valueRow.setColumn(3, new SQLTimestamp(this.getRecordedTime()));
    valueRow.setColumn(4, new SQLVarchar(this.getAssetType()));
    valueRow.setColumn(5, new SQLVarchar(this.getAssetLocation()));
    return valueRow;
}

/**
 * Table definition to use when using a VTI that is an instance of a class
 *
 * @return
 */
public static String getTableDefinition() {
    return "ASSET_NUMBER varchar(50), "
        + "ASSET_DESCRIPTION varchar(100), "
        + "RECORDED_TIME TIMESTAMP, "
        + "ASSET_TYPE VARCHAR(50), "
        + "ASSET_LOCATION VARCHAR(50) ";
}
```

The `getTableDefinition` method is a string description of the table into which you're inserting records; this pretty much replicates the specification you would use in an SQL `CREATE TABLE` statement.

The `getRow` method creates a data row with the appropriate number of columns, uses property getters to set the value of each column, and returns the row as a `resultset` that is compatible with Splice Machine.

About the Sample Code Scripts

These are also two scripts that we use with this tutorial:

Class	Description
<code>/ddl/create-tables.sql</code>	A simple SQL script that you can use to have Splice Machine create the table into which RFID messages are stored.
<code>/scripts/run-mqtt-spark-streaming.sh</code>	Starts the Spark streaming job.

Connecting with Apache Zeppelin

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

This tutorial walks you through connecting your on-premise Splice Machine database with Apache Zeppelin, which is a web-based notebook project currently in incubation at Apache. In this tutorial, you'll learn how to use SQL to query your Splice Machine database from Zeppelin.

NOTE:

Zeppelin is already integrated into the Splice Machine Database-as-Service product; please see our [Using Zeppelin](#) documentation for more information.

See <https://zeppelin.apache.org/> to learn more about Apache Zeppelin.

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

Watch the Video

The following video shows you how to connect Splice Machine with Apache Zeppelin..

Written Walk Through

This section walks you through using SQL to query a Splice Machine database with Apache Zeppelin..

1. Install Zeppelin:

If you're running on AWS, you can install the Zeppelin sandbox application; if you're using an on-premise database, we recommend following the [instructions in this video](#).

2. Create a new interpreter to run with Splice:

- a. Select the **Interpreter** tab in Zeppelin.
- b. Click the **Create** button (in the upper right of the Zeppelin window) to create a new interpreter. Fill in the property fields as follows:

<i>Name</i>	Whatever name you like; we're using SpliceMachine
<i>Interpreter</i>	Select jdbc from the drop-down list of interpreter types.
<i>default.url</i>	jdbc:splice:/myServer:1527/splicedb (replace myServer with the name of the server that you're using)
<i>default password</i>	admin
<i>default userId</i>	splice
<i>common.max_count</i>	1000
<i>default.driver</i>	com.splicemachine.db.jdbc.ClientDriver
<i>Artifacts</i>	Insert the path to the Splice Machine jar file; for example: <div>/tmp/db-client-2.5.0.1708-SNAPSHOT.jar</div>

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:



- 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:

```
%jdbc call MOVIELENS.ContinuousFeatureReport('movielens.user_demographics');
```

The %jdbc specifies that we're creating a paragraph that uses a JDBC interpreter; since we've made the SpliceMachine driver our default JDBC connector, it will be used.

- d. The results of this call look like this:

COLUMN_NAME	MIN	MAX	COUNT	NUM_NONZEROS	STANDARD_DEVIATION	MEAN
AGE	7	73	943	943	12	34

- e. We can also create a new paragraph that performs additional analysis; you'll see that whenever you run a paragraph in Zeppelin, it automatically leaves room at the bottom to create another paragraph.

```
%jdbcselect count(1) num_age, age from MOVIELENS.USER_DEMOGRAPHICS group by age;
```

The results of this paragraph:

```
%jdbc
select count(1) num_age, age from MOVIELENS.USER_DEMOGRAPHICS group by age
```



NUM_AGE	AGE
39	30
23	44
21	40
6	52
3	65
17	34
9	57
3	61
5	13

Took 0 seconds. Last updated by anonymous at time Jul 14, 2016 7:13:56 PM.

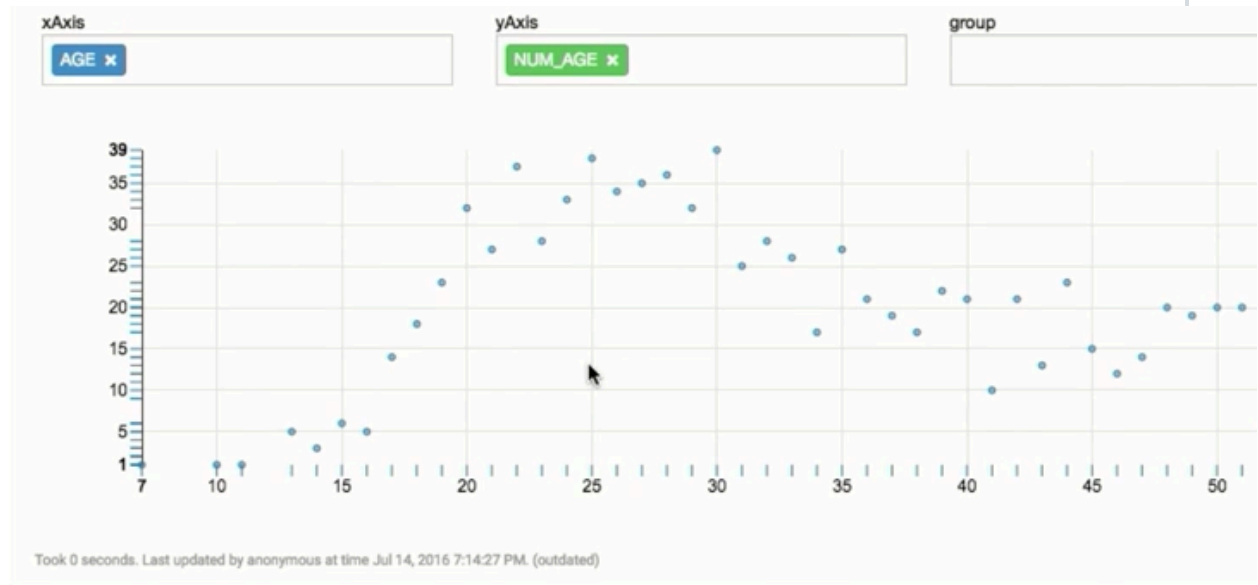
4. Change how you view your data

To get a better sense of what you can do with Zeppelin, we'll modify how we visualize this data:

- 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:



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.](#)
- » [Using HAProxy on a Kerberos-enabled cluster](#)

Configuring HAProxy with Splice Machine

The following example shows you how to configure HAProxy load balancer on a non-Splice Machine node on a Red Hat Enterprise Linux system. Follow these steps:

1. Install HAProxy as superuser :

```
# yum install haproxy
```

2. Configure the `/etc/haproxy/haproxy.cfg` file, following the comments in the sample file below:

In this example, we set the incoming requests to `haproxy_host:1527`, which uses a balancing algorithm of least connections to distribute among the nodes `srv127`, `srv128`, `srv129`, and `srv130`. This means that the incoming connection is routed to the region server that has the least number of connections; thus, the client JDBC URL should point to `<haproxy_host>:1527`.

NOTE: The HAProxy manual describes other balancing algorithms that you can use.

Here is the `haproxy.cfg` file for this example:

```

#-----
# Global settings
#-----
global
    # to have these messages end up in /var/log/haproxy.log you will
    # need to:
    #
    # 1) configure syslog to accept network log events.  This is done
    #    by adding the '-r' option to the SYSLOGD_OPTIONS in
    #    /etc/sysconfig/syslog
    #
    # 2) configure local2 events to go to the /var/log/haproxy.log
    #    file. A line like the following can be added to
    #    /etc/sysconfig/syslog
    #
    #    local2.*                                /var/log/haproxy.log
    #
    maxconn 4000
    log 127.0.0.1 local2
    user haproxy
    group haproxy

#-----
# common defaults that all the 'listen' and 'backend' sections will
# use if not designated in their block
#-----
defaults
    log global
    retries 2
    timeout connect 30000
    timeout server 50000
    timeout client 50000

#-----
# This enables jdbc/odbc applications to connect to HAProxy_host:1527 por
t
# so that HAProxy can balance between the splice engine cluster nodes
# where each node's splice engine instance is listening on port 1527
#-----
listen splice-cluster
    bind *:1527
    log global
    mode tcp
    option tcplog
    option tcp-check
    option log-health-checks
    timeout client 3600s

```



```

timeout server 3600s
balance leastconn
server srv127 10.1.1.227:1527 check
server srv128 10.1.1.228:1527 check
server srv129 10.1.1.229:1527 check
server srv130 10.1.1.230:1527 check

#-----
# (Optional) set up the stats admin page at port 1936
#-----
listen      stats :1936
    mode http
    stats enable
    stats hide-version
    stats show-node
    stats auth admin:password
    stats uri /haproxy?stats

```

Note that some of the parameters may need tuning per the sizing and workload nature:

- » The `maxconnections` parameter indicates how many concurrent connections are served at any given time; you may need to configure this, based on size of the cluster and expected inbound requests.
- » Similarly, the `timeout` values, which are by default in msec, should be tuned so that the connection does not get terminated while a long-running query is executed.

3. Start the HAProxy service:

As superuser, follow these steps to enable the HAProxy service:

Distribution	Instructions
Redhat / CentOS EL6	<pre># chkconfig haproxy on # service haproxy start</pre> <p>If you change the configuration file, reload it with this command:</p> <pre># service haproxy reload</pre>

Distribution	Instructions
Redhat / CentOS EL7	<pre># systemctl enable haproxy ln -s '/usr/lib/systemd/system/haproxy.service' '/etc/systemd/system/multi-user.target.wants/haproxy.service' # systemctl start haproxy</pre> <p>If you change the configuration file, reload it with this command:</p> <pre># systemctl haproxy reload</pre>

NOTE: You can find the HAProxy process id in: `/var/run/haproxy.pid`. If you encounter any issues starting the service, check if Selinux is enabled; you may want to disable it initially.

4. Connect:

You can now connect JDBC clients, including the Splice Machine command line interpreter, `sqlshell.sh`. Use the following JDBC URL:

```
jdbc:splice://<haproxy_host>:1527/splicedb;user=splice;password=admin
```

For ODBC clients to connect through HAProxy, ensure that the DSN entry in file `.odbc.ini` is pointing to the HAProxy host.

5. Verify that inbound requests are being routed correctly:

You can check the logs at `/var/log/haproxy.log` to make sure that inbound requests are being routed to Splice Machine region servers that are receiving inbound requests on port 1527.

6. View traffic statistics:

If you have enabled HAProxy stats, as in our example, you can view the overall traffic statistics in browser at:

```
http://<haproxy_host>:1936/haproxy?stats
```

You'll see a report that looks similar to this:

HAProxy

Statistics Report for pid 751 on stl-colo-10-1-0-236.splicemachine colo

> General process information

pid = 751 (process #1, nbproc = 1)
 uptime = 0s 0h37m00s
 system limits: memmax = unlimited; ulimit-n = 4016
 maxsock = 4016; maxconn = 2000; maxpipes = 0
 current conns = 1001; current pipes = 0; conn rate = 1/sec
 Running tasks: 1/1008; idle = 96 %

active UP
 active UP, going down
 active DOWN, going up
 active or backup DOWN
 active or backup DOWN for maintenance (MAINT)
 backup UP
 backup UP, going down
 backup DOWN, going up
 not checked
 Note: "NOUB/DRAIN" = UP with load-balancing disabled.

Display option:

- Scope :
- Hide DOWN's
- Disable refresh
- Refresh now
- CSV export

splice-cluster		Queue		Session rate		Sessions						Bytes		Denied		Errors		Warnings		Serve						
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act
Frontend					0	116	-	1 000	1 000	2 000	3 001			1 244 674	70 579 321	0	0	0					OPEN			
srv127		0	0	-	0	29		250	250	-	751	751	5m14s	473 674	69 991 321	0	0	0	0	0	0	0	37m UP	L4OK in 0ms	1	Y
srv128		0	0	-	0	29		250	250	-	750	750	5m14s	257 000	196 000	0	0	0	0	0	0	0	37m UP	L4OK in 0ms	1	Y
srv129		0	0	-	0	29		250	250	-	750	750	5m14s	257 000	196 000	0	0	0	0	0	0	0	37m UP	L4OK in 0ms	1	Y
srv130		0	0	-	0	29		250	250	-	750	750	5m14s	257 000	196 000	0	0	0	0	0	0	0	37m UP	L4OK in 0ms	1	Y
Backend		0	0		0	116		1 000	1 000	200	3 001	3 001	5m14s	1 244 674	70 579 321	0	0		0	0	0	0	37m UP		4	4

stats		Queue		Session rate		Sessions						Bytes		Denied		Errors		Warnings		Serve						
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act
Frontend					1	4	-	1	3	2 000	85			61 755	1 118 907	0	0	12					OPEN			
Backend		0	0		0	2		0	1	200	71	0	0s	61 755	1 118 907	0	0		71	0	0	0	37m UP		0	0

Using HAProxy with Splice Machine on a Kerberos-Enabled Cluster

Your JDBC and ODBC applications can authenticate to the backend region server through HAProxy on a Splice Machine cluster that has Kerberos enabled,

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

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

Kerberos and Application Access

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

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

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

Kerberos and ODBC Access

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

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

```
USE_KERBEROS=1
```

See our [Using the Splice Machine ODBC Driver](#) for more information.

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

```
kinit principal
```

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

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

Example

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

1. Create the principal in Kerberos Key Distribution Center

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

2. Copy the generated keytab file

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

3. Connect:

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

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

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

1. Create the user in your Splice Machine database:

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

2. Grant privileges to the user

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

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

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

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

4. Set the password for the new principal:

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

5. Create keytab file jdoe.keytab

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

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

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

Connecting to Splice Machine with Java and JDBC

This topic shows you how to compile and run a sample Java program that connects to Splice Machine using our JDBC driver. The `SampleJDBC` program does the following:

- » connects to a standalone (`localhost`) version of Splice Machine
- » creates a table named `MYTESTTABLE`
- » inserts several sample records
- » issues a query to retrieve those records

[Follow the written directions below](#), which includes the raw code for the `SampleJDBC` example program.

Compile and Run the Sample Program

This section walks you through compiling and running the `SampleJDBC` example program, in the following steps:

1. Locate the Splice Machine JDBC Driver:

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

```
/splicemachine/jdbc-driver/
```

- or -

```
/splicemachine/jdbc-driver/db-client-2.5.0.1734.jar
```

NOTE: The build number, e.g. 1729, varies with each Splice Machine software update.

2. Copy the example program code:

You can copy and paste the code below:

```

package com.splicemachine.cs.tools

import java.sql.*;

/**
 * Simple example that establishes a connection with splice and does a few
 * basic JDBC operations
 */
public class SampleJDBC {
    public static void main(String[] arg) {
        //JDBC Connection String - sample connects to local database
        String dbUrl = "jdbc:splice://localhost:1527/splicedb;user=splice;password=admin";

        try{
            //For the JDBC Driver - Use the Apache Derby Client Driver
            Class.forName("com.splicemachine.db.jdbc.ClientDriver");
        }catch(ClassNotFoundException cne){
            cne.printStackTrace();
            return; //exit early if we can't find the driver
        }

        try(Connection conn = DriverManager.getConnection(dbUrl)){
            //Create a statement
            try(Statement statement = conn.createStatement()){

                //Create a table
                statement.execute("CREATE TABLE MYTESTTABLE(a int, b varchar(30))");

                //Insert data into the table
                statement.execute("insert into MYTESTTABLE values (1,'a')");
                statement.execute("insert into MYTESTTABLE values (2,'b')");
                statement.execute("insert into MYTESTTABLE values (3,'c')");
                statement.execute("insert into MYTESTTABLE values (4,'c')");
                statement.execute("insert into MYTESTTABLE values (5,'c')");

                int counter=0;
                //Execute a Query
                try(ResultSet rs=statement.executeQuery("select a, b from MYTESTTABLE")){
                    while(rs.next()){
                        counter++;
                        int val_a=rs.getInt(1);
                        String val_b=rs.getString(2);
                        System.out.println("record=["+counter+"] a=["+val_a+"] b=["+val_b+"]");
                    }
                }
            }
        }
    }
}

```

```

    }

    }catch (SQLException se) {
        se.printStackTrace();
    }
}

```

Note that the code uses the default JDBC URL and driver class values:

Connection Parameter	Default Value	Comments
JDBC URL	<code>jdbc:splice://<hostname>:1527/splicedb</code>	<p>Use localhost as the <code><hostname></code> value for the standalone version of Splice Machine.</p> <p>On a cluster, specify the IP address of an HBase RegionServer.</p>
JDBC driver class	<code>com.splicemachine.db.jdbc.ClientDriver</code>	

3. Compile the code

Compile and package the code into `splice-jdbc-test-0.1.0-SNAPSHOT.jar`.

4. Run the program:

When you run the program, your `CLASSPATH` must include the path to the Splice Machine JDBC driver. If you did compile and package your code into `splice-jdbc-test-0.1.0-SNAPSHOT.jar`, you can run the program with this command line:

```
java -cp splice-installer-<platformVersion>/resources/jdbc-driver/ com.splicemachine.cs.tools.SampleJDBC
```

The command should display a result like the following:

```
record=[1] a=[1] b=[a]
record=[2] a=[2] b=[b]
record=[3] a=[3] b=[c]
record=[4] a=[4] b=[c]
record=[5] a=[5] b=[c]
```


Connecting to Splice Machine with JRuby and JDBC

This topic shows you how to compile and run a sample JRuby program that connects to Splice Machine using our JDBC driver. The JRubyJDBC program does the following:

- » connects to a standalone (localhost) version of Splice Machine
- » selects and displays the records in a table

Compile and Run the Sample Program

This section walks you through compiling and running the JRubyJDBC example program, in the following steps:

1. Install the JDBC Adapter gem

Use the following command to install the `activerecord-jdbcd Derby-adapter` gem:

```
gem install activerecord-jdbcd Derby-adapter
```

2. Configure the connection

You must assign the database connectivity parameters in the `config/database.yml` file for your JRuby application. Your connectivity parameters should look like the following, which use our default database, user, URL, and password values:

```
# Configure Using Gemfile
# gem 'activerecord-jdbcd sqlite3-adapter'
# gem 'activerecord-jdbcd Derby-adapter'
#
development:
  adapter: jdbcd Derby
  database: splicedb
  username: splice
  password: admin
  driver: com.splicemachine.db.jdbc.ClientDriver
  url: jdbc:splice://localhost:1527/splicedb
```



Use `localhost:1527` with the standalone (local computer) version of `splice machine`. If you're running `Splice Machine` on a cluster, substitute the address of your server for `localhost`; for example:

```
jdbc:splice://mySrv123cba:1527/splicedb.
```

3. Create the sample data table

Create the `MYTESTTABLE` table in your database and add a little test data. Your table should look something like the following:

```
splice> describe SPLICE.MYTESTTABLE;
COLUMN_NAME          |TYPE_NAME|DEC  |NUM  |COLUMN|COLUMN_DEF|CHAR_OCTE  |I
S_NULL
-----
-----
A                      |INTEGER  |0     |10   |10     |NULL       |NULL       |YES
B                      |VARCHAR  |NULL  |NULL |30     |NULL       |60         |YES

2 rows selected

splice> select * from MYTESTTABLE order by A;
A      |B
-----
1      |a
2      |b
3      |c
4      |c
5      |c

5 rows selected
```

4. Copy the code

You can copy the example program code and paste it into your editor:

```

require 'java'

module JavaLang
  include_package "java.lang"
end

module JavaSql
  include_package 'java.sql'
end

import 'com.splicemachine.db.jdbc.ClientDriver'

begin
  conn = JavaSql::DriverManager.getConnection("jdbc:splice://localhost:1527/splicedb;user=splice;password=admin");
  stmt = conn.createStatement
  rs = stmt.executeQuery("select a, b from MYTESTTABLE")
  counter = 0
  while (rs.next) do
    counter+=1
    puts "Record=[" + counter.to_s + "] a=[" + rs.getInt("a").to_s +
    "]" b=[" + rs.getString("b") + "]"
  end
  rs.close
  stmt.close
  conn.close()
  rescue JavaLang::ClassNotFoundException => e
    stderr.print "Java told me: #{e}\n"
  rescue JavaSql::SQLException => e
    stderr.print "Java told me: #{e}\n"
end

```

5. Run the program

Run the JRubyConnect program as follows

```
jruby jrubbyjdbc.rb
```

The command should display a result like the following:

```

Record=[1] a=[3] b=[c]
Record=[2] a=[4] b=[c]
Record=[3] a=[5] b=[c]
Record=[4] a=[1] b=[a]
Record=[5] a=[2] b=[b]

```

Connecting to Splice Machine with Jython and JDBC

This topic shows you how to compile and run a sample Jython program that connects to Splice Machine using our JDBC driver. The `print_tables` program does the following:

- » connects to a standalone (`localhost`) version of Splice Machine
- » selects and displays records from one of the system tables

Compile and Run the Sample Program

This section walks you through compiling and running the `print_tables` example program, in the following steps:

1. **Add the Splice client jar to your `CLASSPATH`; for example:**

```
export CLASSPATH=/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):

```

from java.sql import DriverManager
from java.lang import Class
from java.util import Properties

url      = 'jdbc:splice://localhost:1527/splicedb'
driver   = 'com.splicemachine.db.jdbc.ClientDriver'
props    = Properties()
props.setProperty('user', 'splice')
props.setProperty('password', 'admin')
jcc      = Class.forName(driver).newInstance()
conn     = DriverManager.getConnection(url, props)
stmt     = conn.createStatement()
rs       = stmt.executeQuery("select * from sys.systables")

rowCount = 0
while (rs.next() and rowCount < 10) :
    rowCount += 1
    print "Record=[" + str(rowCount) + "]"
        id   = [" + rs.getString('TABLEID') + "]
        name = [" + rs.getString('TABLENAME') + "]
        type = [" + rs.getString('TABLETYPE') + "]"

rs.close()
stmt.close()
conn.close()

```

3. Save the code to `print_files.jy`.

4. Run the program:

Run the `print_tables.jy` program as follows:

```
jython print_tables.jy
```

The command should display a result like the following:

```
Record=[1] id=[f9f140e7-0144-fcd8-d703-00003cbfba48] name=[ASDAS] type=[T]
Record=[2] id=[c934c123-0144-fcd8-d703-00003cbfba48] name=[DDC_NOTMAILABLE] type=[T]
Record=[3] id=[dd5cc163-0144-fcd8-d703-00003cbfba48] name=[FRANK_EMCONT_20130430] type=[T]
Record=[4] id=[f584c1a3-0144-fcd8-d703-00003cbfba48] name=[INACTIVE_QA] type=[T]
Record=[5] id=[cfcc41df-0144-fcd8-d703-00003cbfba48] name=[NUM_GROOM4] type=[T]
Record=[6] id=[6b7f4217-0144-fcd8-d703-00003cbfba48] name=[PP_LL_QA] type=[T]
Record=[7] id=[ac154287-0144-fcd8-d703-00003cbfba48] name=[QUAL_BRANDS] type=[T]
Record=[8] id=[d67d42c7-0144-fcd8-d703-00003cbfba48] name=[SCIENCE_CAT_QA] type=[T]
Record=[9] id=[c1e0c303-0144-fcd8-d703-00003cbfba48] name=[TESTOUTPUT2] type=[T]
Record=[10] id=[f408c343-0144-fcd8-d703-00003cbfba48] name=[UAC_1267_AVJ] type=[T]
```

Connecting to Splice Machine with Scala and JDBC

This topic shows you how to compile and run a sample Scala program that connects to Splice Machine using our JDBC driver. The `SampleScalaJDBC` program does the following:

- » connects to a standalone (`localhost`) version of Splice Machine
- » creates a table named `MYTESTTABLE`
- » inserts several sample records
- » selects and displays records from one of the system tables

Compile and Run the Sample Program

This section walks you through compiling and running the `SampleScalaJDBC` example program, in the following steps:

1. **Add the Splice client jar to your `CLASSPATH`; for example:**

```
export CLASSPATH=/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 s
 * tatements
 */
object SampleScalaJDBC{

    def main(args: Array[String]) {

        // connect to the database named "splicedb" on the localh
        ost
        val driver = "com.splicemachine.db.jdbc.ClientDriver"
        val dbUrl = "jdbc:splice://localhost:1527/splicedb;user=s
        plice;password=admin"

        var connection:Connection = null

        try {

            // make the connection
            Class.forName(driver)
            connection = DriverManager.getConnection(dbUrl)

            // create the statement
            var statement = connection.createStatement()

            //Create a table
            statement.execute("CREATE TABLE MYTESTTABLE(a in
            t, b varchar(30))");

            statement.close

            //Insert data into the table
            var pst = connection.prepareStatement("insert int
            o MYTESTTABLE (a,b) values (?,?)")

            pst.setInt (1, 1)
            pst.setString (2, "a")
            pst.executeUpdate()

            pst.clearParameters()
            pst.setInt (1, 2)
            pst.setString (2, "b")
            pst.executeUpdate()

```



```

        pst.clearParameters()
        pst.setInt (1, 3)
        pst.setString (2, "c")
        pst.executeUpdate()

        pst.clearParameters()
        pst.setInt (1, 4)
        pst.setString (2, "c")
        pst.executeUpdate()

        pst.clearParameters()
        pst.setInt (1, 5)
        pst.setString (2, "c")
        pst.executeUpdate()

        pst.close

        //Read the data
        statement = connection.createStatement()
        val resultSet = statement.executeQuery("select
a, b from MYTESTTABLE")
        var counter =0

        while ( resultSet.next() ) {
            counter += 1
            val val_a = resultSet.getInt(1)
            val val_b = resultSet.getString(2)
            println("record=[" + counter + "] a=[" +
val_a + "] b=[" +val_b + "]")
        }
        resultSet.close()
        statement.close()
    } catch {
        case ex : java.sql.SQLException => println("SQLEx
ception: "+ex)
    } finally {
        connection.close()
    }
}
}

```

3. Save the code to `SampleScalaJDBC.scala`.

4. Compile the program:

```
scalac SampleScalaJDBC.scala
```

5. Run the program:

Run the `SampleScalaJDBC` program as follows:

```
scala SampleScalaJDBC
```

The command should display a result like the following:

```
record=[1] a=[5] b=[c]  
record=[2] a=[1] b=[a]  
record=[3] a=[2] b=[b]  
record=[4] a=[3] b=[c]  
record=[5] a=[4] b=[c]
```

Connecting to Splice Machine with NodeJS / AngularJS

This topic shows you how to connect to Splice Machine with NodeJS and AngularJS.

Watch the Video

The following video shows you how to connect NodeJS and Angularjs with Splice Machine.

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:

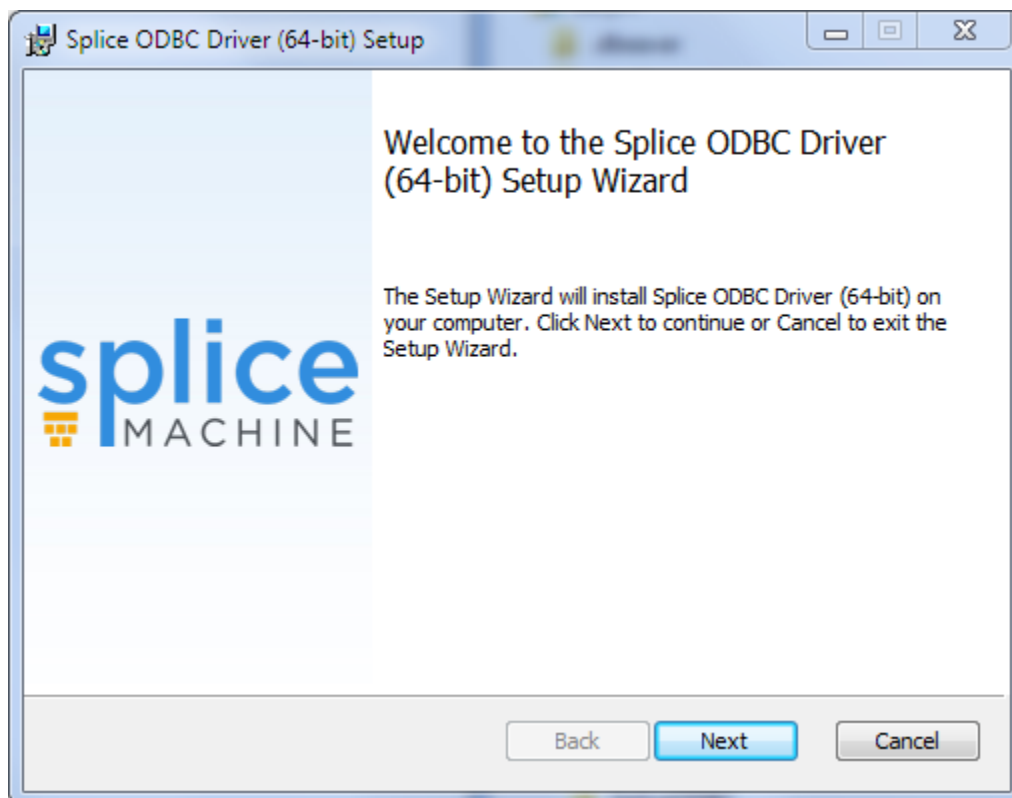
The file you download will have a name similar to these:

» 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:

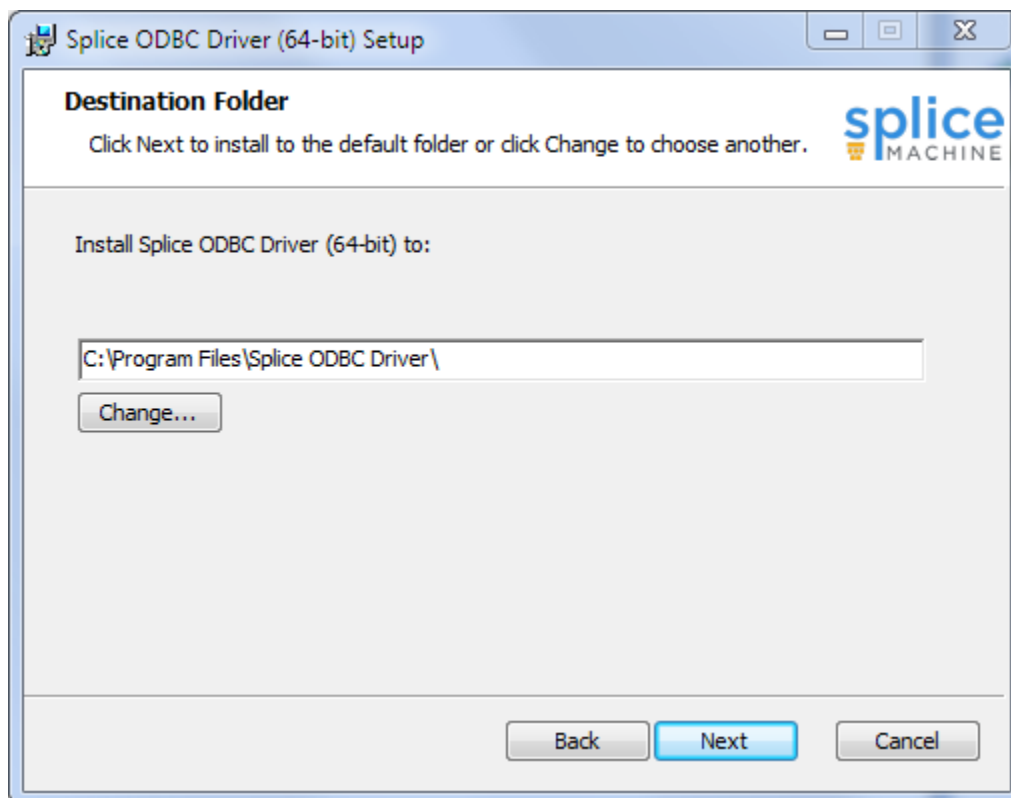


Click the **Next** button to proceed.

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.

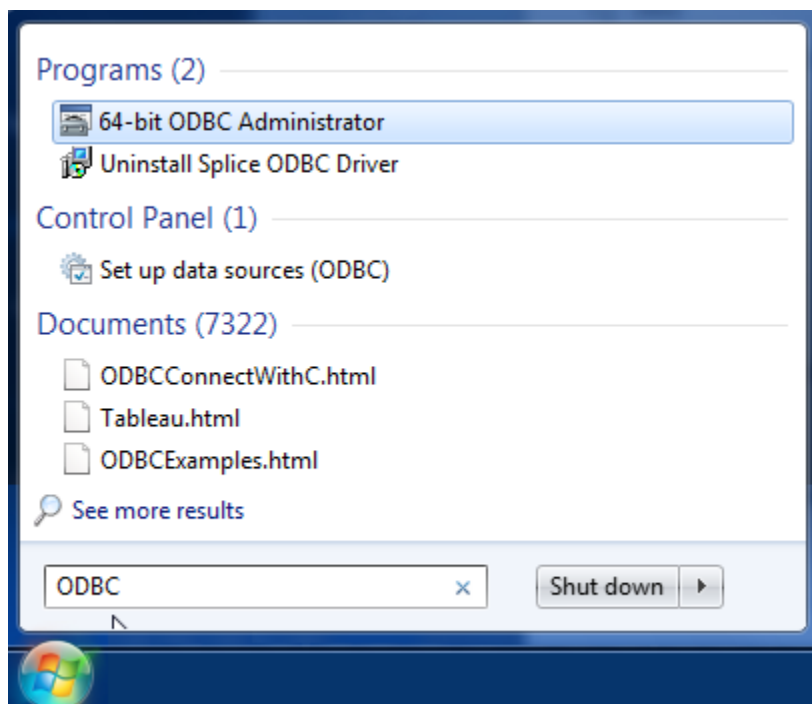
6. Finish the installation

Click the **Finish** button, and you're ready to use the Splice Machine ODBC driver.

7. Start the Windows ODBC Data Source Administrator tool

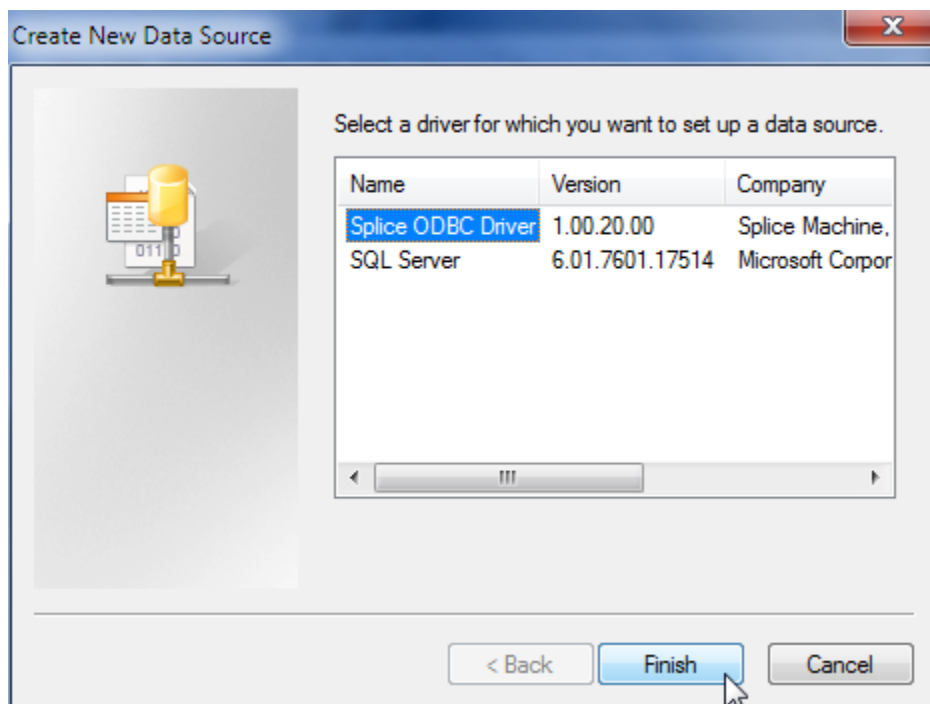
You need to add our ODBC driver to the set of Windows ODBC data sources, using the Windows ODBC Data Source Administrator tool; You can read about this tool here: [https://msdn.microsoft.com/en-us/library/ms712362\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms712362(v=vs.85).aspx).

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

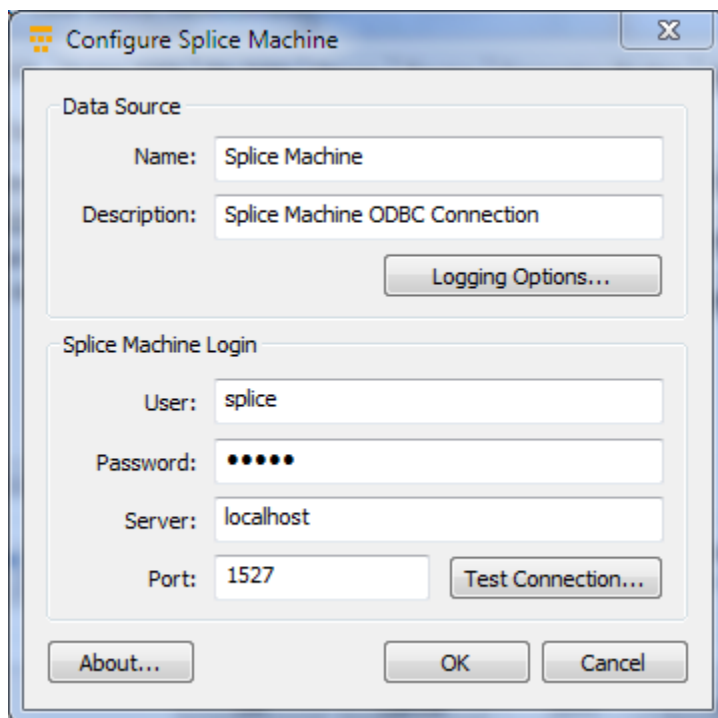
Click the **Add** button on the User DSN tab of the ODBC Data Source Administrator screen, and then select the Splice Machine driver you just installed:



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.

Set the fields in the *Data Source* and *Splice Machine Login* sections similarly to the settings shown here:



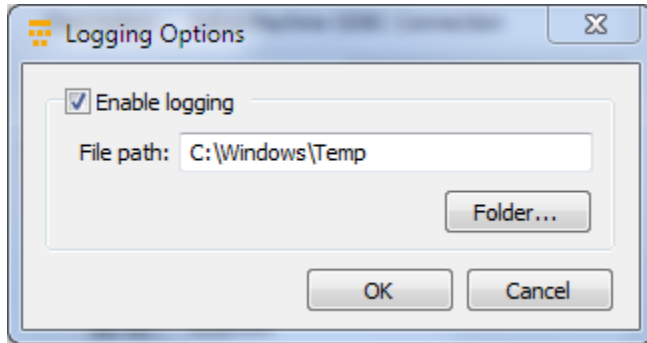
The default user name is `splice`, and the default password is `admin`.

NOTE: For `Server`: on a cluster, specify the IP address of an HBase RegionServer. If you're running the standalone version of Splice Machine, specify `localhost`.

If you have Splice Machine running, you can click the *Test...* button at the bottom of the Configuration dialog to verify that all is well.

10. Configure logging (optional):

You can optionally configure the ODBC driver to log activity. This can be handy for debugging connection issues; however, it adds overhead and will have a significant impact on performance. Click the `Logging Options` button in the ODBC Administrator *Configuration* screen to enable or disable logging:



Installing the Driver on Linux

Follow these steps to install the Splice Machine ODBC driver on a Linux computer:

1. Make sure you have unixODBC installed.

You must have version 2.2.12 or later of the `unixODBC` driver manager installed to run the Splice Machine ODBC driver.

Some Linux distributions include `unixODBC`, while others do not. Our driver will not work without it. For more information about `unixODBC`, see: <http://www.unixodbc.org>.

2. Download the installer:

You can download the driver installer from our ODBC download site: <https://www.splicemachine.com/get-started/odbc-driver-download/>

Download the installer to the Linux computer on which you want to install the driver. The file will have a name similar to this:

```
splice_odbc_64-1.0.28.0.x86_64.tar.gz
```

3. Unzip the installation package

Use the following command to unpack the tarball you installed, substituting in the actual version number from the download:

```
tar xzf splice_odbc_64-<version>.x86_64.tar.gz
```

This creates a directory named `splice_odbc_64-<version>`.

4. Install the driver:

Navigate to the directory that was created when you unzipped the tarball, and run the install script:

If you run the script as root, the default installation directory is `/usr/local/splice`:

```
sudo ./install.sh
```

If you run the script as a different user, the driver is installed to `~/splice`.

```
./install.sh
```

The script creates a `splice` directory in the install location; you'll be prompted for that location, which defaults to `/usr/local`.

You'll also be prompted to enter the IP address of the Splice Machine server, which defaults to `127.0.0.1`.

The install directory, e.g. `/usr/local/splice`, will contain two subdirectories:

Directory	Contents
<code>lib64</code>	The driver binary.
<code>errormessages</code>	The XML error message source for any error messages issued by the driver.

5. Configure the driver:

If you ran the installation script as root, the `odbc.ini`, `odbcinst.ini`, and `splice.odbcdriver.ini` configuration files were copied into the `/etc` folder, and any previous copies were renamed, e.g. `odbc.ini.1`.

If you did not run the installation script as root, then hidden versions of the same files are located in your `$HOME` directory: `.odbc.ini`, `.odbcinst.ini`, and `.splice.odbcdriver.ini`.

File	Description
<code>odbc.ini</code>	Specifies the ODBC data sources (DSNs).
<code>odbcinst.ini</code>	Specifies the ODBC drivers.
<code>splice.odbcdriver.ini</code>	Configuration information specific to the Splice Machine ODBC driver.

The default version of the `odbc.ini` file looks like this:

```

[ODBC Data Sources]
SpliceODBC64           = SpliceODBCDriver

[SpliceODBC64]
Description            = Splice Machine ODBC 64-bit
Driver                 = /usr/local/splice/lib64/libsplice_odbc.so
UID                    = splice
PWD                    = admin
URL                    = 127.0.0.1
PORT                   = 1527
SSL                    = peerAuthentication
SSL_CERT               = /home/splice/client.pem
SSL_PKEY               = /home/splice/client.key
SSL_TRUST              = TRUE

```

If you specified a different installation directory, you need to update the `Driver` location setting in your `odbc.ini` file. This is not typically required; however, if you do make this change, you should copy your modified file to the `/etc` directory.

```
cp odbc.ini /etc/
```

If you are connecting to a Kerberos-enabled cluster using ODBC, you **must add this parameter**:

```
USE_KERBEROS          = 1
```

For more information about connecting to a Kerberos-enabled cluster, see [Connecting to Splice Machine Through HAProxy](#).

6. Configure Driver Logging, if desired


You can edit the `splice.odbcdriver.ini` file to configure driver logging, which is disabled by default:

```

[Driver]
DriverManagerEncoding=UTF-16
DriverLocale=en-US
ErrorMessagesPath=/usr/local/splice/errormessages/
LogLevel=0
LogNamespace=
LogPath=
ODBCInstLib=/usr/lib64/libodbcinst.so

```

To configure logging, modify the `LogLevel` and `LogPath` values:

LogLevel	<p>You can specify one of the following values:</p> <pre> 0 = OFF 1 = LOG_FATAL 2 = LOG_ERROR 3 = LOG_WARNING 4 = LOG_INFO 5 = LOG_DEBUG 6 = LOG_TRACE </pre> <p>The larger the LogLevel value, the more verbose the logging.</p> <div>  <p>Logging does impact driver performance.</p> </div>
LogPath	<p>The path to the directory in which you want the logging files stored. Two log files are written in this directory:</p> <pre> 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. </pre>

After configuring logging, copy the file to `/etc`:

7. Verify your installation

You can test your installation by using the following command to run `isql`:

```
isql SpliceODBC64 splice admin
```

Installing the Driver on MacOS

Follow these steps to install the Splice Machine ODBC driver on a MacOS computer:



Our MacOS ODBC driver is currently in Beta release.

1. Make sure you have iODBC installed.

You must have an ODBC administration driver to manage ODBC data sources on your Mac. We recommend installing the iODBC driver for the Mac, which you'll find on the iODBC site: www.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/>

Download the installer to the MacOS computer on which you want to install the driver. The file will have a name similar to this:

```
splice_odbc_64_macosx64-2.5-45.0.tar.gz
```

3. Unzip the installation package

Use the following command to unpack the tarball you installed, substituting in the actual version number from the download:

```
tar -xzf splice_odbc_64_macosx64-<version>.tar.gz
```

This creates a directory named `splice_odbc_macosx64`.

4. Install the driver:

Navigate to the directory that was created when you unzipped the tarball, and run the install script:

```
./install.sh
```

Follow the installer prompts. In most cases, you can simply accept the default values.

The installer will create several files in the install directory, including these three files, which contain the configuration info that can be modified as required:

File	Description
<code>odbc.ini</code>	Specifies the ODBC data sources (DSNs).
<code>odbcinst.ini</code>	Specifies the ODBC drivers.
<code>splice.odbcdriver.ini</code>	Configuration information specific to the Splice Machine ODBC driver.

If you have not previously installed our ODBC driver, the installer will also copy the files into `$HOME/Library/ODBC` for use with iODBC.

5. Configure the driver:

The installed driver is configured with settings that you specified when responding to the installer prompts. You can change values as follows:

a. Edit the `odbc.ini` file to match your configuration.

You'll find the `odbc.ini` file in your `$HOME/Library/ODBC` directory; we also create a link to this file in `$HOME/.odbc.ini`. You can edit `odbc.ini` (or `.odbc.ini`) from either location.

NOTE: The `URL` field in the `odbc.ini` file is actually the IP address of the Splice Machine server.

The default version of the `odbc.ini` file looks like this:

```
[ODBC Data Sources]
SpliceODBC64          = SpliceODBCDriver

[SpliceODBC64]
Description           = Splice Machine ODBC 64-bit
Driver                = /usr/local/splice/lib64/libsplice_odbc.so
UID                   = splice
PWD                   = admin
URL                   = 0.0.0.0
PORT                  = 1527
```

If you are connecting to a Kerberos-enabled cluster using ODBC, you **must add this parameter**:

```
USE_KERBEROS          = 1
```

For more information about connecting to a Kerberos-enabled cluster, see [Connecting to Splice Machine Through HAProxy](#).

b. Edit (if desired) and copy the `splice.odbcdriver.ini` file:


The `splice.odbcdriver.ini` file contains information specific to the driver. You can edit this file to configure driver logging, which is disabled by default:

```
[Driver]
DriverManagerEncoding=UTF-16
DriverLocale=en-US
ErrorMessagesPath=/usr/local/splice/errormessages/
LogLevel=0
LogNamespace=
LogPath=
ODBCInstLib=/usr/lib64/libodbcinst.so
```

A copy of the Splice Machine ODBC configuration file, `splice.odbcdriver.ini`, which contains the default values, was copy to `/Library/ODBC/SpliceMachine` during installation. You will need root access to modify this file:

```
sudo vi /Library/ODBC/SpliceMachine/splice.odbcdriver.ini
```

To configure logging, modify the `LogLevel` and `LogPath` values:

LogLevel	<p>You can specify one of the following values:</p> <pre> 0 = OFF 1 = LOG_FATAL 2 = LOG_ERROR 3 = LOG_WARNING 4 = LOG_INFO 5 = LOG_DEBUG 6 = LOG_TRACE </pre> <p>The larger the <code>LogLevel</code> value, the more verbose the logging.</p> <div>  Logging does impact driver performance. </div>
LogPath	<p>The path to the directory in which you want the logging files stored. Two log files are written in this directory:</p> <pre> 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 </pre>

6. Verify your installation

You can test your installation by launching the 64-bit version of the iODBC Data Source Administrator for both configuring and testing your DSNs. Note that you can also perform your `odbc.ini` modifications with this tool instead of manually editing the file.

Using the ODBC Driver with C

This section contains a simple example of using the Splice Machine ODBC driver with the C programming language. This program simply displays information about the installed driver. You can compile and run it by following these steps:

1. Copy the code

You can copy and paste the code below:

```
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>

main() {
    SQLHENV env;
    char driver[256];
    char attr[256];
    SQLSMALLINT driver_ret;
    SQLSMALLINT attr_ret;
    SQLUSMALLINT direction;
    SQLRETURN ret;

    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
    SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION, (void *) SQL_OV_ODBC3, 0);

    direction = SQL_FETCH_FIRST;
    while(SQL_SUCCEEDED(ret = SQLDrivers(env, direction,
        driver, sizeof(driver), &driver_ret,
        attr, sizeof(attr), &attr_ret))) {
        direction = SQL_FETCH_NEXT;
        printf("%s - %s\n", driver, attr);
        if (ret == SQL_SUCCESS_WITH_INFO) printf("\tdata truncation\n");
    }
}
```

2. Compile it

```
#!/bin/bash
# gcc -I /usr/local/splice/unixODBC/include listODBCdriver.c -o listODBCdriver -L/usr/local/splice/lib -lodbc -lodbcinst -lodbccr
```

3. Run the program

Run the compiled listODBCdriver:

```
prompt:~$ ./listODBCdriver
```

The command should display a result like the following:

```
Splice Machine - Description=Splice Machine ODBC Driver
```


Connecting to Splice Machine with Python and ODBC

This topic shows you how to compile and run a sample Python program that connects to Splice Machine using our ODBC driver. The `SpliceODBCConnect.py` program does the following:

- » connects to a standalone (`localhost`) version of Splice Machine
- » retrieves and displays records from several system tables
- » creates a table inserts several sample records into it
- » selects and aggregates records from the new table

Compile and Run the Sample Program

This section walks you through compiling and running the `SpliceODBCConnect.py` example program, in the following steps:

1. Install the Splice Machine ODBC driver

[Follow our instructions](#) for installing the driver on Unix or Windows.

2. Install the pyodbc module

You need to install the `pyodbc` open source Python module, which implements the DB API 2.0 specification and can be used with Python 2.4 or higher. See <https://github.com/mkleehammer/pyodbc> for more information about this module.

To install `pyodbc` on the server on which you'll be running your job:

```
yum install gcc-c++pip install pyodbc
```

3. Confirm that you can connect

To confirm that you're ready to use the ODBC driver, launch the python shell and enter the following commands, replacing `SpliceODBC64` with the name of your data source (which is found in the `odbc.ini` file that you edited when installing our ODBC driver):

```
import pyodbc
cnxn = pyodbc.connect("DSN=SpliceODBC64")
cursor = cnxn.cursor()
cursor.execute("select * from SYS.SYSTABLES")
row = cursor.fetchone()
print('row:', row)
```

4. Copy the example program code:

You can copy and paste the code below:

```
#!/usr/bin/python

# This program is used to demonstrate connecting to Splice Machine using
ODBC

import pyodbc

#Connect to Splice Machine using an Datasource
cnxn = pyodbc.connect("DSN=SpliceODBC64")

#Open a cursor
cursor = cnxn.cursor()

#Build a select statement
cursor.execute("select * from SYS.SYSTABLES")

#Fetch one record from the select
row = cursor.fetchone()

#If there is a record, print it
if row:
    print(row)

#The following will continue to retrieve one record at a time from the re
sultset
while 1:
    row = cursor.fetchone()
    if not row:
        break
    print('table name:', row.TABLENAME)

#The following is an example of using the fetchall option, instead of ret
rieving one record at time
cursor.execute("select * from SYS.SYSSCHEMAS")
rows = cursor.fetchall()
for row in rows:
    print(row.SCHEMAID, row.SCHEMANAME)

#Create a table
cursor.execute("CREATE TABLE MYPYTHONTABLE(a int, b varchar(30))")

#Insert data into the table
cursor.execute("insert into MYPYTHONTABLE values (1,'a')");
cursor.execute("insert into MYPYTHONTABLE values (2,'b')");
cursor.execute("insert into MYPYTHONTABLE values (3,'c')");
cursor.execute("insert into MYPYTHONTABLE values (4,'c')");
cursor.execute("insert into MYPYTHONTABLE values (5,'c')");
```

```
#Commit the creation of the table
cnxn.commit();

#Confirm the records are in the table
row = cursor.execute("select count(1) as TOTAL from SPLICE.MYPYTHONTABLE").fetchone()
print(row.TOTAL)
```

5. Save the code to `SpliceODBCConnect.py`.

6. Run the program:

Run the `SpliceODBCConnect.py` program as follows:

```
python ./SpliceODBCConnect.py
```

Connecting to Splice Machine with C and ODBC

This topic shows you how to compile and run a sample C program that exercises the Splice Machine ODBC driver. The `listODBCdriver` program verifies that the driver is correctly installed and available.

Compile and Run the Sample Program

This section walks you through compiling and running the `listODBCdriver` example program, which simply displays information about the installed driver.

1. Install the ODBC driver

[Follow our instructions](#) for installing the driver on Unix or Windows.

2. Copy the example program code

You can copy and paste the code below:

```
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>

main() {
    SQLHENV env;
    char driver[256];
    char attr[256];
    SQLSMALLINT driver_ret;
    SQLSMALLINT attr_ret;
    SQLUSMALLINT direction;
    SQLRETURN ret;

    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
    SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION, (void *) SQL_OV_ODBC3, 0);

    direction = SQL_FETCH_FIRST;
    while(SQL_SUCCEEDED(ret = SQLDrivers(env, direction,
        driver, sizeof(driver), &driver_ret,
        attr, sizeof(attr), &attr_ret))) {
        direction = SQL_FETCH_NEXT;
        printf("%s - %s\n", driver, attr);
        if (ret == SQL_SUCCESS_WITH_INFO) printf("\tdata truncation\n");
    }
}
```

3. Compile it

```
#!/bin/bash
# gcc -I /usr/local/splice/unixODBC/include listODBCdriver.c -o listODBCdriver -L/usr/local/splice/lib -lodbcc -lodbccinst -lodbccr
```

4. Run the program

Run the compiled `listODBCdriver`:

```
prompt:~$ ./listODBCdriver
```

The command should display a result like the following:

```
Splice Machine - Description=Splice Machine ODBC Driver
```

Connecting DBeaver with Splice Machine Using JDBC

This topic shows you how to connect DBeaver to Splice Machine using our JDBC driver. To complete this tutorial, you need to:

- » Have Splice Machine installed and running on your computer.
- » Have DBeaver installed on your computer. You can download an installer and find directions on the DBeaver web site (dbeaver.jkiss.org).

Connect DBeaver with Splice Machine

This section walks you through configuring DBeaver to connect with Splice Machine

1. **Install DBeaver , if you've not already done so**

[Follow the instructions on the DBeaver web site.](#)

2. **Start a Splice Machine session on the computer on which you have installed DBeaver**

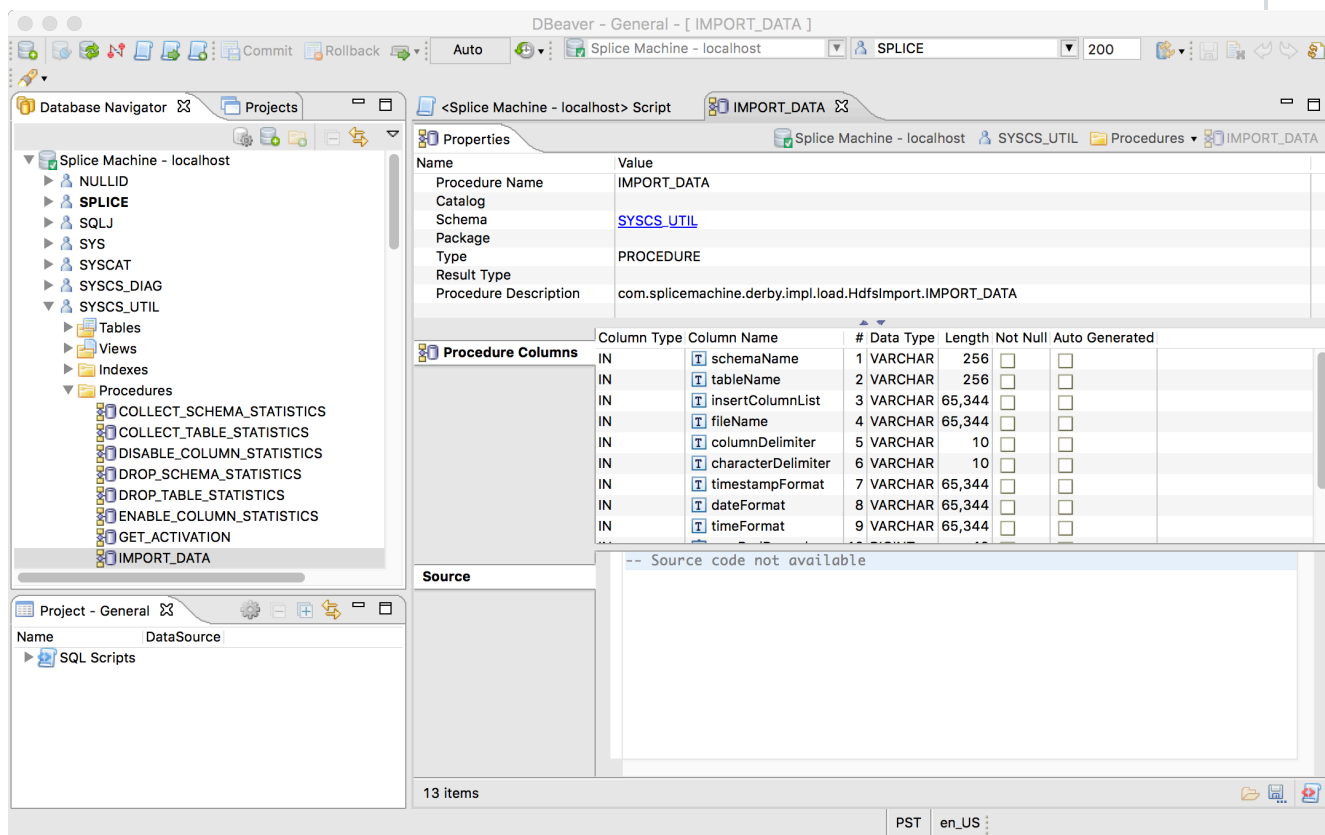
Splice Machine must be running to create and use it with DBeaver .

3. **Configure a Splice Machine connection in DBeaver**

Follow the instructions in the next section, [Configure a DBeaver Connection for Splice Machine](#), to create and test a new connection in DBeaver .

4. **Connect DBeaver to Splice Machine**

In DBeaver's *Database Navigator*, select the Splice Machine connection you configured. Your database will display, and you can inspect objects or enter SQL to interact with your data.



Configure a DBeaver Connection for Splice Machine

Follow these steps to configure and test a new driver and connection alias in DBeaver .

1. Start a Splice Machine session on the computer on which you have installed DBeaver
2. Open the DBeaver application.
3. **Select `Driver Manager` in the DBeaver `Database` menu, then click the `New` button to create a new driver:**
 - a. Specify values in the `Create New Driver` form; these are the default values:

Field	Value
Driver Name:	Any name you choose
Class Name:	<code>com.splicemachine.db.jdbc.ClientDriver</code>
URL Template:	<code>jdbc:splice://{host}:{port}/splicedb;create=false</code>
Default Port:	1527
Description:	Any description you want to specify

- b.** Click the `Add File` button, then navigate to and select the Splice JDBC Driver jar file. which you'll find it in the `jdbc-driver` folder under the `splicemachine` directory on your computer.

Create new driver

Settings

Driver Name*: Driver Type:

Class Name*:

URL Template:

Default Port: ☐ Embedded

Description

Category:

Description:

Libraries **Connection properties** **Adv. parameters**

Add File

Add Folder

Add Artifact

Download/Update

Information

Delete

Classpath

Driver class: Find Class

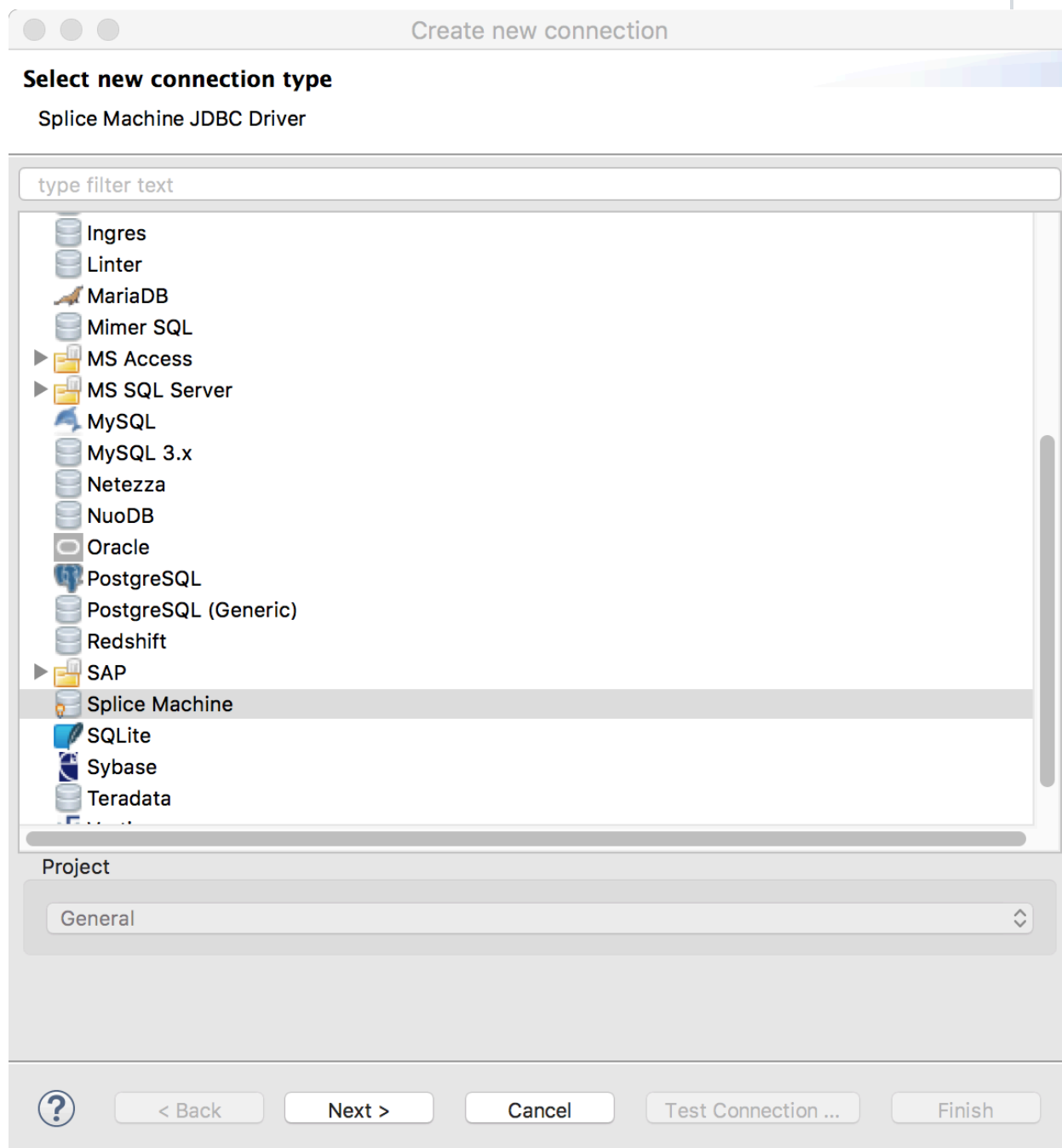
? Reset to Defaults Cancel OK

NOTE: Instead of manually entering the `Class Name` for your driver, you can click the `Find Class` button to discover the driver class name associated with the file you've located.

- Click **OK** to save the driver entry and close the form.

5. Select **New Connection** in the DBeaver *Database* menu, then follow these steps to create a new connection that uses our driver:

- a. Scroll through the connection type list and select the Splice Machine JDBC driver that you just created, then click the **Next** button:



- b. Several of the fields in the *Generic JDBC Connection Settings* screen were pre-populated for you when you selected the driver. You need to fill in the **User name:** (default is `splice`) and **Password:** (default is `admin`) field values:

Create new connection

Generic JDBC Connection Settings
Splice Machine connection settings

General Driver properties

JDBC URL: jdbc:splice://localhost:1527/splicedb;create=false

Host: localhost Port: 1527

User name: splice

Password: •••••

Driver Name: Splice Machine Edit Driver Settings

? < Back Next > Cancel Test Connection ... Finish

- c. Click the **Test Connection** button to verify your connection.

NOTE: Splice Machine must be running on your computer for the connection test to succeed.

- d. Click the **Next** button to reveal the network configuration screen. If you have VPN requirements, enter the appropriate information in this screen; if not, simply click the **Next** button again.

The screenshot shows a macOS-style window titled "Create new connection". Inside, the "Network" section is active, with the subtitle "Configure networks handlers and tunnels". There are two tabs: "SSH Tunnel" (selected) and "SOCKS Proxy". Under the "SSH Tunnel" tab, there is a checkbox labeled "Use SSH Tunnel" which is currently unchecked. Below this are four fields: "Host/IP:" (a text input field), "Port:" (a spinner box showing "22"), "User Name:" (a text input field), and "Authentication Method:" (a dropdown menu showing "Password"). At the bottom of the window, there is a row of buttons: a help icon (question mark in a circle), "< Back", "Next >", "Cancel", "Test Connection ...", and "Finish".

- e. You can optionally modify any settings in the *Finish connection creation* screen; then click the `Finish` button to save your new connection.

● ● ●

Create new connection

Finish connection creation
General connection settings.

Connection name:

Connection type:

Connection folder:

Security

☒ Save password locally

Miscellaneous

☒ Show system objects
☐ Show utility objects

Connection

Auto-commit: ☒

Isolation level:

Default schema:

Keep-Alive:

Connecting DBVisualizer with Splice Machine Using JDBC

This topic shows you how to connect DBVisualizer to Splice Machine using our JDBC driver. To complete this tutorial, you need to:

- » Have Splice Machine installed and running on your computer.
- » Have DBVisualizer installed on your computer. You can find directions on the DBVisualizer web site (<https://www.dbvis.com>); you can also download a free trial version of DBVisualizer from there.

Connect DBVisualizer with Splice Machine

This section walks you through configuring DBVisualizer to connect with Splice Machine

- 1. Install DBVisualizer, if you've not already done so**

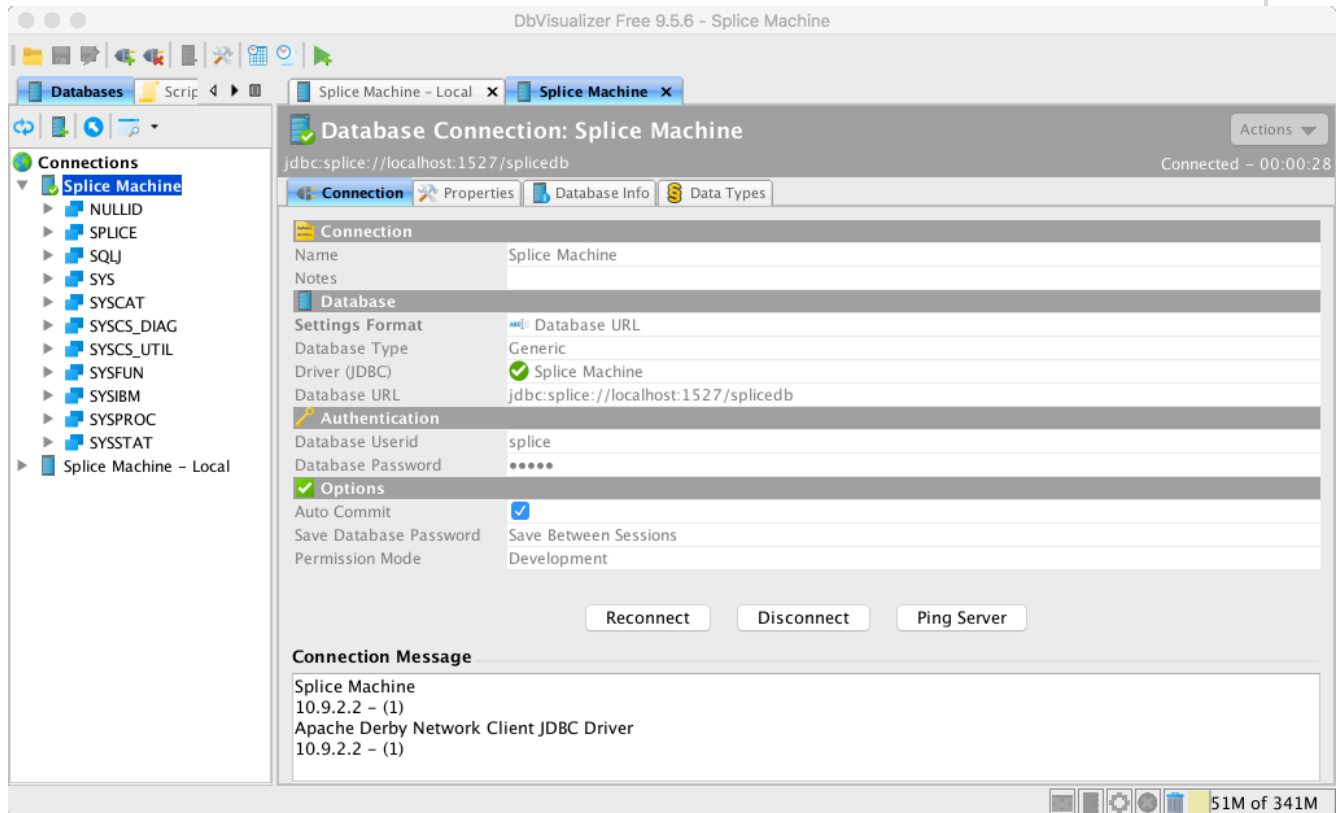
[Follow the instructions on the DBVis web site.](#)

- 2. Configure a Splice Machine connection in DBVisualizer**

Follow the instructions in the next section, [Configure a DBVisualizer Connection for Splice Machine](#), to create and test a new connection in DBVisualizer.

- 3. Connect DBVisualizer to Splice Machine**

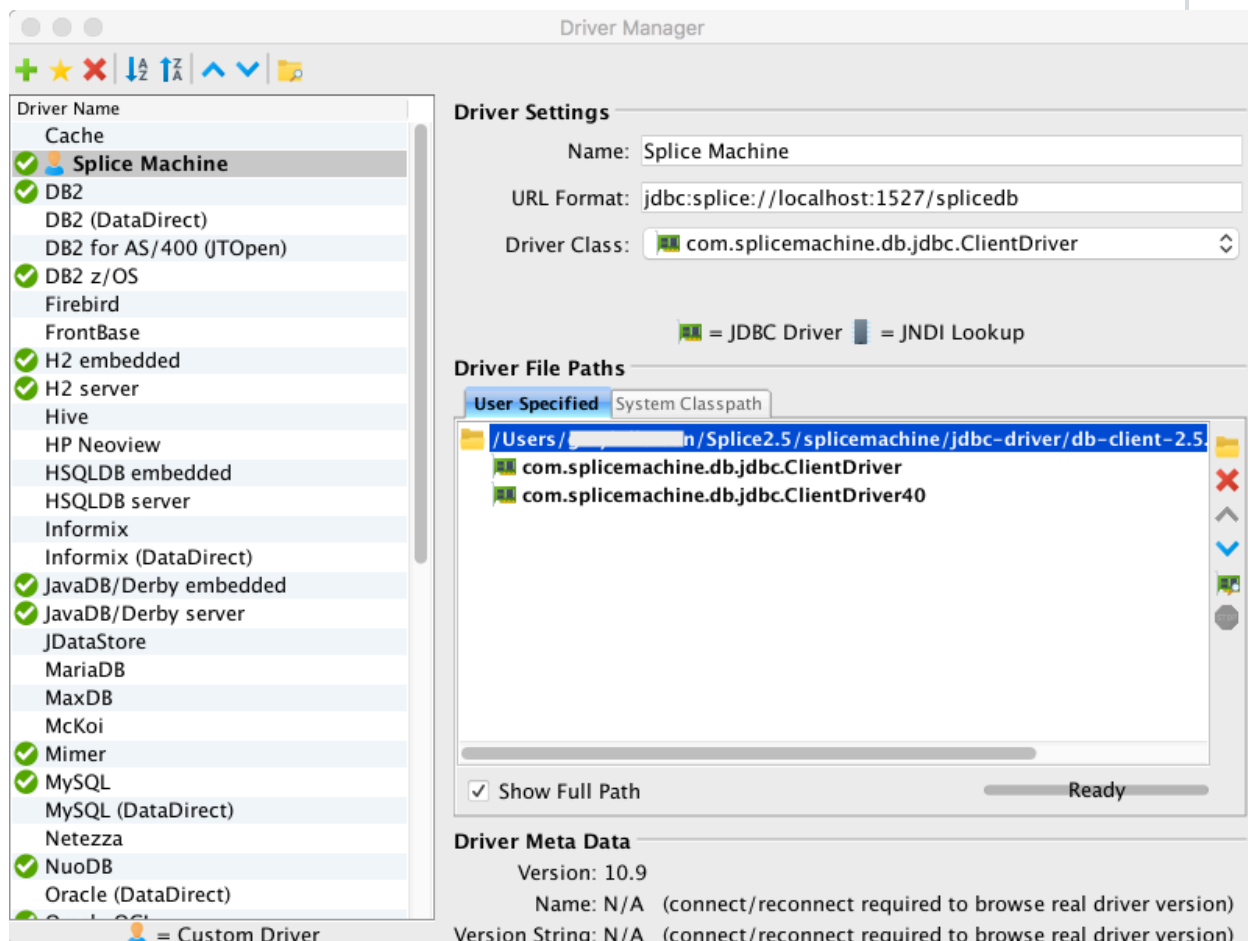
In DBVisualizer, open the connection alias you created and click the **Connect** button. Your database will display in DBVisualizer, and you can inspect objects or enter SQL to interact with your data.



Configure a DBVisualizer Connection for Splice Machine

Follow these steps to configure and test a new driver entry and connection in DBVisualizer.

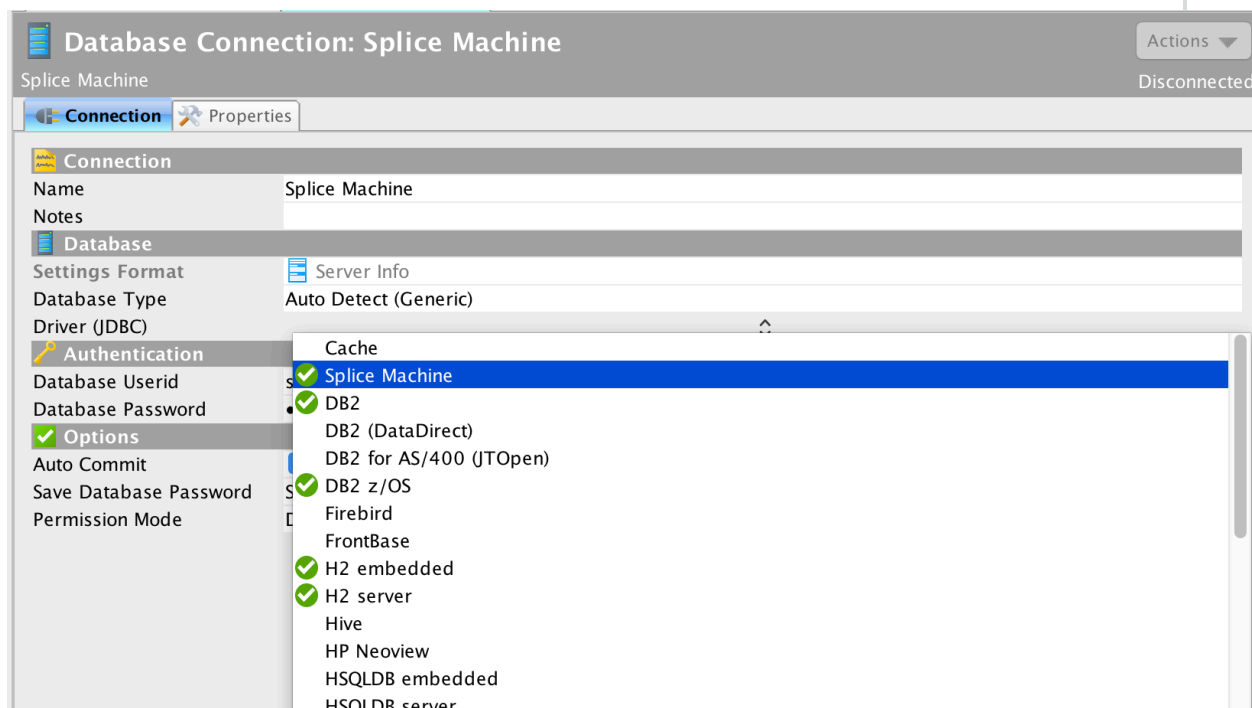
1. Start a Splice Machine session on the computer on which you have installed DBVisualizer.
2. Open the DBVisualizer application.
3. **Use the Driver Manager to create a new DBVisualizer driver entry.**
 Select **Driver Manager** from the **Tools** menu; in the **Driver Manager** screen:
 - a. Click the green plus sign + button to add a new driver entry.
 - b. Name the driver and enter `jdbc:splice://localhost:1527/splicedb` in the **URL Format** field:



- c. In the Driver File Paths section, click `User Specified`, and then click the yellow folder icon.
- d. Navigate to and select the Splice JDBC Driver jar file. which you'll find it in the `jdbc-driver` folder under the `splicemachine` directory on your computer.
- e. Close the Driver Manager screen.

4. Create a DBVisualizer connection alias that uses the new driver:

- a. Select `Create Database Connection` from the *Database* menu. If prompted about using the Wizard, click the `No Wizard` button.
- b. Name the connection (we use `Splice Machine`), then click the empty field next to the `Driver (JDBC)` caption and select the driver you just created:



- 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** (`splice`) and **Password** (`admin`) fields. Then click the **Connect** button. Your Splice Machine database will now display in DBVisualizer:

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 SQuirreL with Splice Machine Using JDBC

This topic shows you how to connect SQuirreL to Splice Machine using our JDBC driver. To complete this tutorial, you need to:

- » Have Splice Machine installed and running on your computer.
- » Have SQuirreL installed on your computer. You can find directions on the SQuirreL web site (<http://squirrel-sql.sourceforge.net/>); you can also download a free trial version of SQuirreL from there. You must also install the Derby plug-in for SQuirreL.

Connect SQuirreL with Splice Machine

This section walks you through configuring SQuirreL to connect with Splice Machine

- 1. Install SQuirreL, if you've not already done so:**

[Follow the instructions on the SQuirreL web site.](http://squirrel-sql.sourceforge.net/)

- 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**

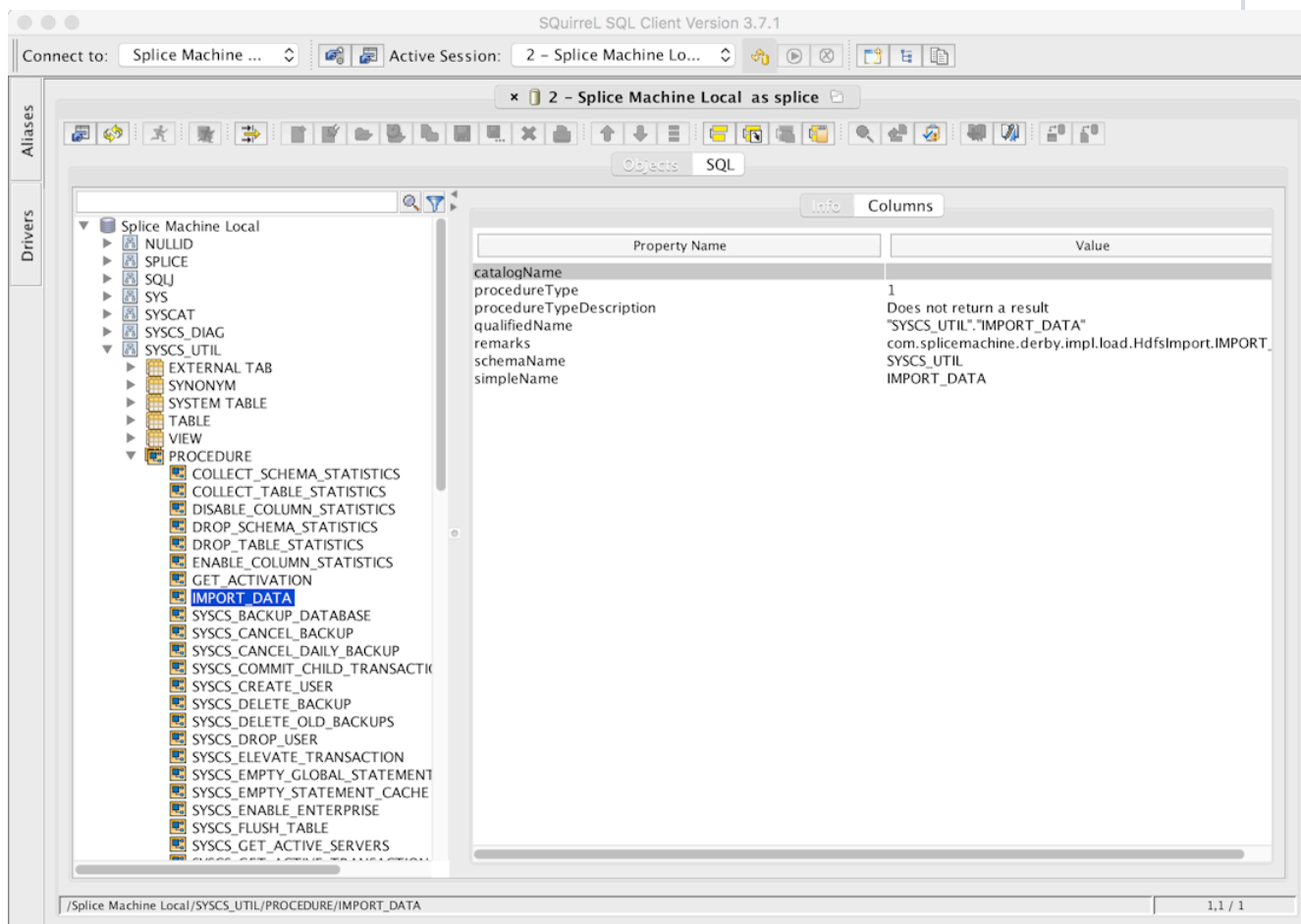
Splice Machine must be running to create and use it with SQuirreL.

- 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**

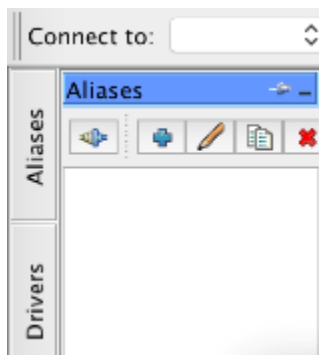
In SQuirreL, open the connection alias you created, enter your credentials, and click the `Connect` button. Your database will display in SQuirreL, and you can inspect objects or enter SQL to interact with your data.



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:

Change Driver: Splice Machine

Add Driver

Driver

Name:

Example URL:

Website URL:

Java Class Path Extra Class Path

List Drivers

Up

Down

Add

Delete

Class Name:

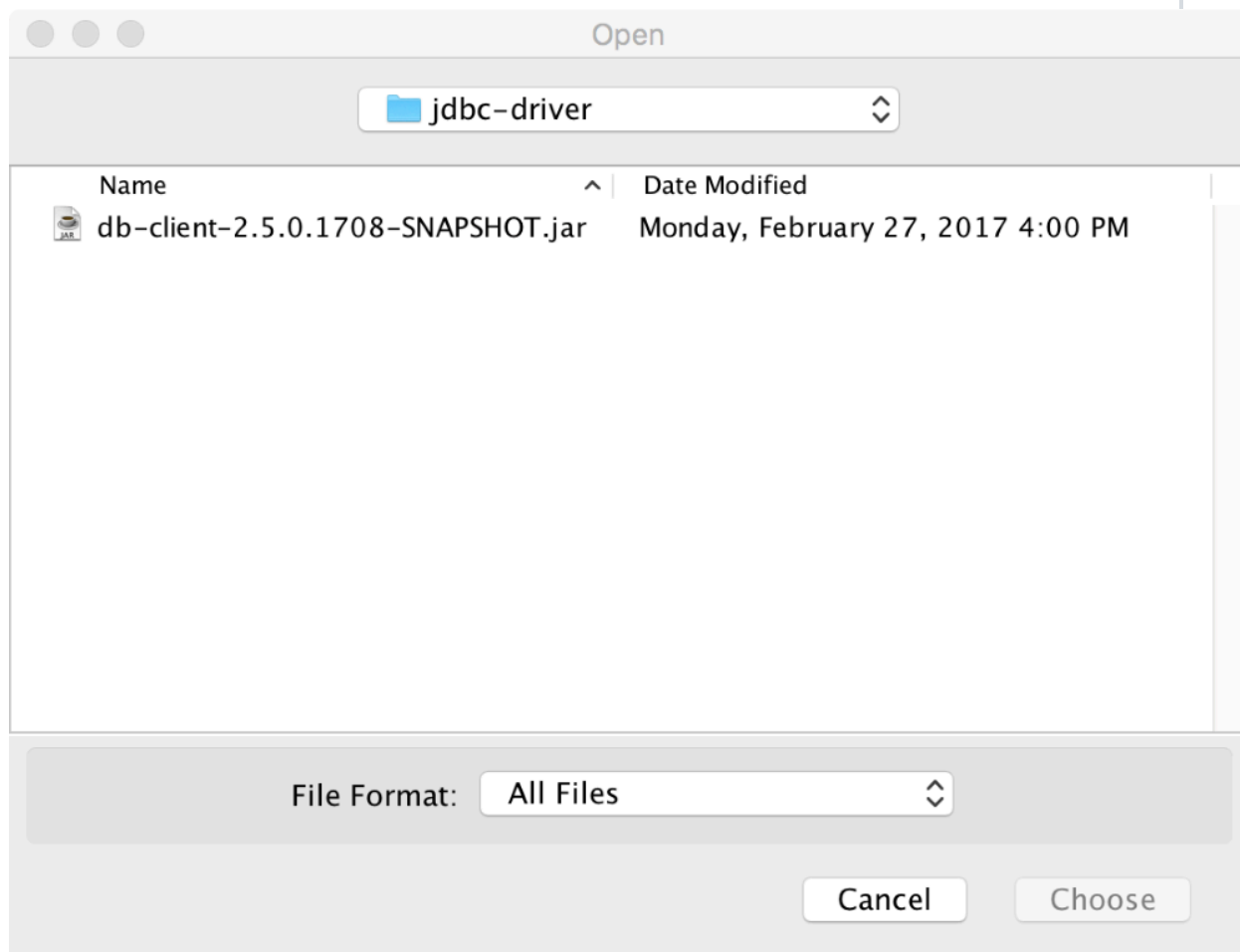
OK Close



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.`

- b. Click the Extra Class Path button, and click the Add button.

- c. Navigate to and select the Splice JDBC Driver jar file. which you'll find it in the `jdbc-driver` folder under the `splicemachine` directory on your computer.



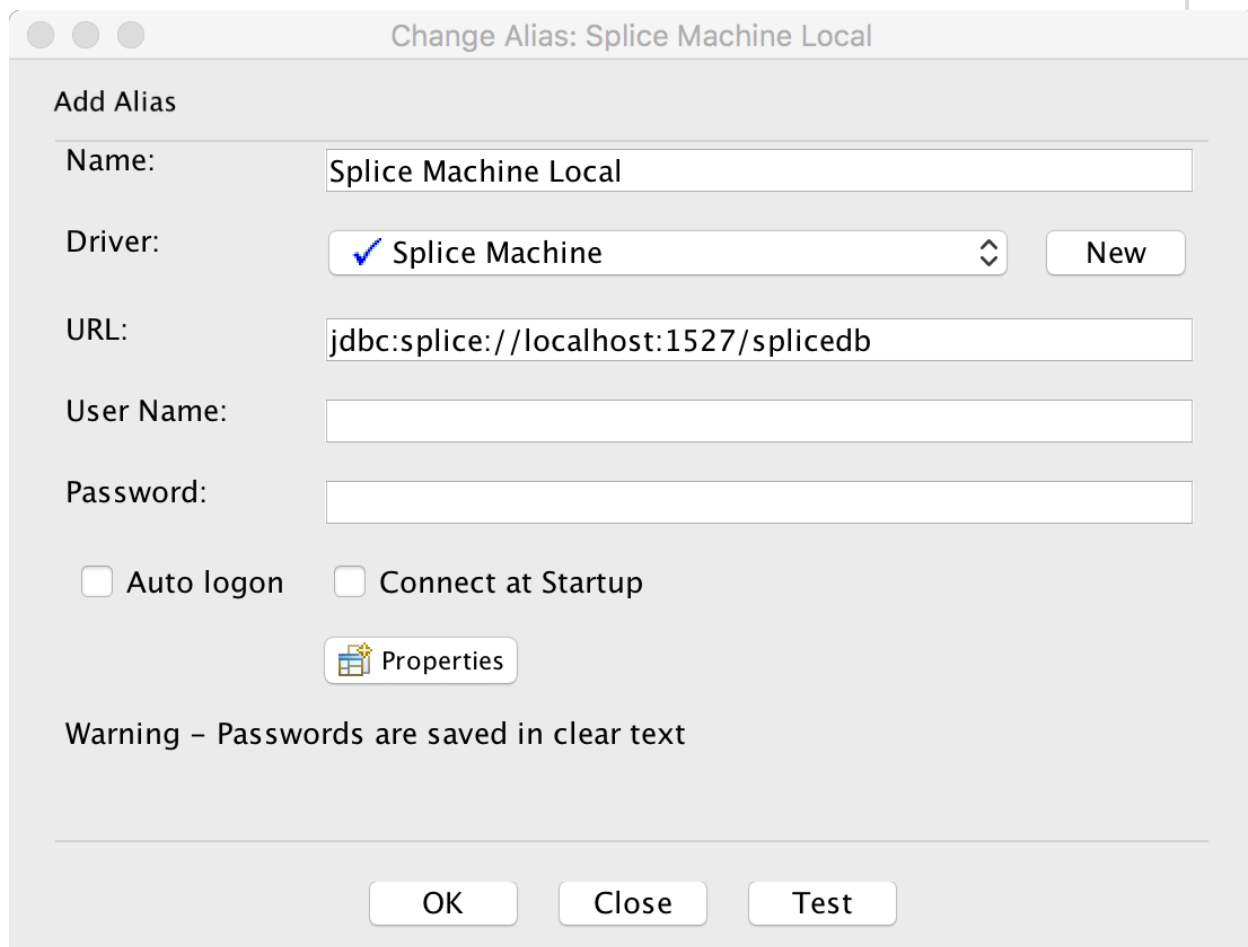
- d. Now, back in the `Add Driver` screen, click the `List Drivers` button verify that you see the Splice Machine driver:

```
com.splicemachine.db.jdbc.ClientDriver
```

- e. Click the OK button to add the driver entry in SquirrelL.

5. Create a connection alias in SquirrelL

- Click the *Aliases* tab in the SquirrelL window, and then click the `Create new Alias` (blue + sign) button.
- Enter a name for your alias and select the driver you just created from the drop-down list



Change Alias: Splice Machine Local

Add Alias

Name: Splice Machine Local


Driver: ✓ Splice Machine ↕ New

URL: jdbc:splice://localhost:1527/splicedb

User Name:

Password:

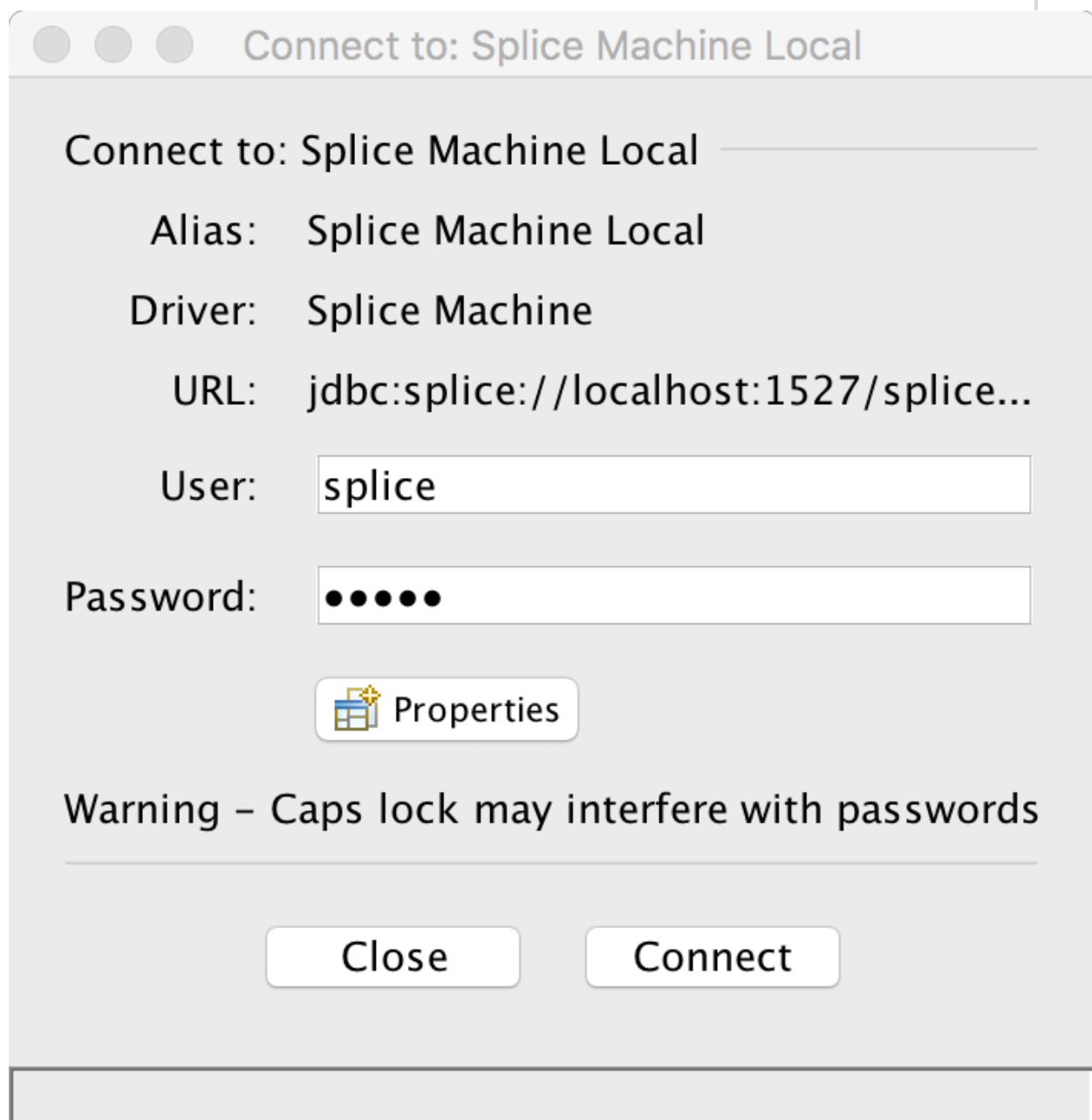
☐ Auto logon ☐ Connect at Startup

 Properties

Warning - Passwords are saved in clear text

OK Close Test

- c. Click the `Test` button to verify your connection. In the Connect screen, enter `splice` as the `User:` value and `admin` for the `Password:` value.



Connect to: Splice Machine Local


Alias: Splice Machine Local

Driver: Splice Machine

URL: jdbc:splice://localhost:1527/splice...

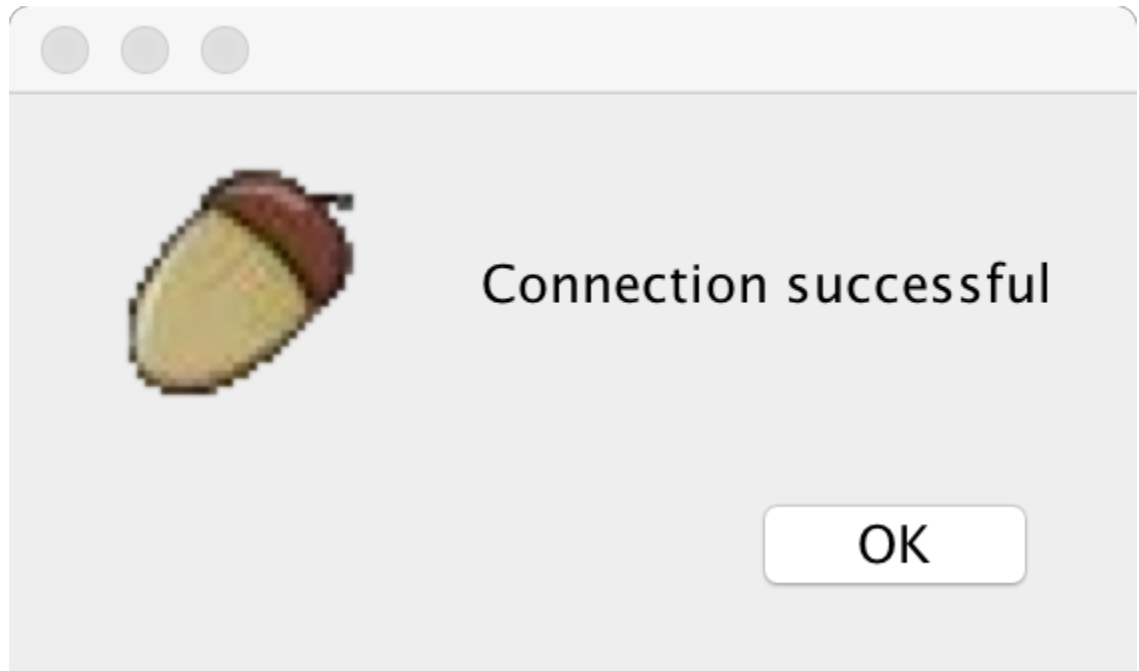
User:

Password:

 Properties

Warning – Caps lock may interfere with passwords

- d. Click the `Connect` button to verify your connection. You should see the success message:



Connecting Tableau with Splice Machine Using ODBC

This topic shows you how to connect Tableau to Splice Machine using our ODBC driver. To complete this tutorial, you need to:

- » Have Tableau installed on your Windows or MacOS computer. You can find directions on the Tableau web site (www.tableau.com); you can also download a free trial version of Tableau from there.
- » Have the Splice Machine ODBC driver installed on your computer. Follow the instructions in our Developer's Guide.

Connect Tableau with Splice Machine

This section walks you through configuring Tableau on a Windows PC to connect with Splice Machine using our ODBC driver.

1. Install Tableau, if you've not already done so

[Follow the instructions on the Tableau web site.](http://www.tableau.com)

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.