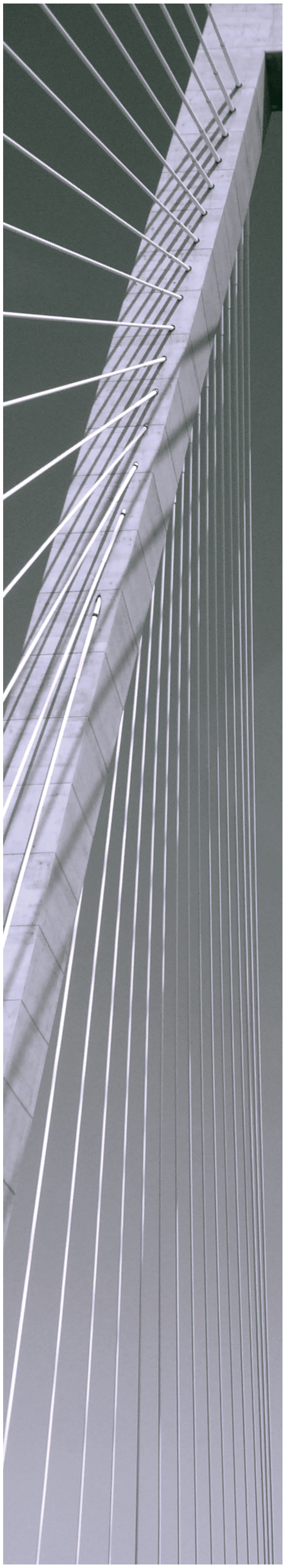

Simba Cassandra JDBC Driver with SQL Connector

Installation and Configuration Guide

Simba Technologies Inc.

Version 2.0.4

November 29, 2019

Contact Us

Simba Technologies Inc.
938 West 8th Avenue
Vancouver, BC Canada
V5Z 1E5

Tel: +1 (604) 633-0008

Fax: +1 (604) 633-0004

www.simba.com

About This Guide

Purpose

The *Simba Cassandra JDBC Driver with SQL Connector Installation and Configuration Guide* explains how to install and configure the Simba Cassandra JDBC Driver with SQL Connector on all supported platforms. The guide also provides details related to features of the driver.

Audience

The guide is intended for end users of the Simba Cassandra JDBC Driver.

Knowledge Prerequisites

To use the Simba Cassandra JDBC Driver, the following knowledge is helpful:

- Familiarity with the platform on which you are using the Simba Cassandra JDBC Driver
- Ability to use the data store to which the Simba Cassandra JDBC Driver is connecting
- An understanding of the role of JDBC technologies in connecting to a data store
- Experience creating and configuring JDBC connections
- Exposure to SQL

Document Conventions

Italics are used when referring to book and document titles.

Bold is used in procedures for graphical user interface elements that a user clicks and text that a user types.

`Monospace font` indicates commands, source code or contents of text files.

Note:

A text box with a pencil icon indicates a short note appended to a paragraph.

! **Important:**

A text box with an exclamation mark indicates an important comment related to the preceding paragraph.

Table of Contents

About the Simba Cassandra JDBC Driver

The Simba Cassandra JDBC Driver enables Business Intelligence (BI), analytics, and reporting on data that is stored in Apache Cassandra databases, including support for databases hosted on the cloud as DataStax Astra databases. The driver complies with the JDBC 4.0, 4.1, and 4.2 data standards.

Note:

Support for JDBC 4.0 is deprecated, and will be removed in a future release of this driver. For more information, see the release notes.

JDBC is one of the most established and widely supported APIs for connecting to and working with databases. At the heart of the technology is the JDBC driver, which connects an application to the database. For more information about JDBC, see *Data Access Standards* on the Simba Technologies website: https://www.simba.com/resources/data-access-standards-glossary.

This guide is suitable for users who want to access data residing within Cassandra from their desktop environment. Application developers might also find the information helpful. Refer to your application for details on connecting via JDBC.

System Requirements

Each machine where you use the Simba Cassandra JDBC Driver must have Java Runtime Environment (JRE) 7.0 or 8.0 installed. If you are using the driver with JDBC API version 4.2, or connecting to DataStax Astra databases, then you must use JRE 8.0.

The driver supports Apache Cassandra versions 2.1 through 3.11.

Simba Cassandra JDBC Driver Files

The Simba Cassandra JDBC Driver is delivered in the following ZIP archives, where *[Version]* is the version number of the driver:

- `CassandraJDBC4_[Version].zip`
- `CassandraJDBC41_[Version].zip`
- `CassandraJDBC42_[Version].zip`

The archive contains the driver supporting the JDBC API version indicated in the archive name, as well as release notes and third-party license information. In addition, the required third-party libraries and dependencies are packaged and shared in the driver JAR file in the archive.

Note:

Support for JDBC 4.0 is deprecated, and will be removed in a future release of this driver. For more information, see the release notes.

Installing and Using the Simba Cassandra JDBC Driver

To install the Simba Cassandra JDBC Driver on your machine, extract the files from the appropriate ZIP archive to the directory of your choice.

! **Important:**

If you received a license file through email, then you must copy the file into the same directory as the driver JAR file before you can use the Simba Cassandra JDBC Driver.

To access a Cassandra data store using the Simba Cassandra JDBC Driver, you need to configure the following:

- The list of driver library files (see Referencing the JDBC Driver Libraries on page 10)
- The `Driver` or `DataSource` class (see Registering the Driver Class on page 11)
- The connection URL for the driver (see Building the Connection URL on page 12)

The Simba Cassandra JDBC Driver provides read-write access to Cassandra data.

Referencing the JDBC Driver Libraries

Before you use the Simba Cassandra JDBC Driver, the JDBC application or Java code that you are using to connect to your data must be able to access the driver JAR files. In the application or code, specify all the JAR files that you extracted from the ZIP archive.

Using the Driver in a JDBC Application

Most JDBC applications provide a set of configuration options for adding a list of driver library files. Use the provided options to include all the JAR files from the ZIP archive as part of the driver configuration in the application. For more information, see the documentation for your JDBC application.

Using the Driver in Java Code

You must include all the driver library files in the class path. This is the path that the Java Runtime Environment searches for classes and other resource files. For more information, see "Setting the Class Path" in the appropriate Java SE Documentation.

For Java SE 7:

- For Windows:
  http://docs.oracle.com/javase/7/docs/technotes/tools/windows/classpath.html
- For Linux and Solaris:
  http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/classpath.html

For Java SE 8:

- For Windows:
  http://docs.oracle.com/javase/8/docs/technotes/tools/windows/classpath.html
- For Linux and Solaris:
  http://docs.oracle.com/javase/8/docs/technotes/tools/solaris/classpath.html

Registering the Driver Class

Before connecting to your data, you must register the appropriate class for your application.

The following classes are used to connect the Simba Cassandra JDBC Driver to Cassandra data stores:

- The `Driver` classes extend `java.sql.Driver`.
- The `DataSource` classes extend `javax.sql.DataSource` and `javax.sql.ConnectionPoolDataSource`.

To support JDBC 4.0, classes with the following fully-qualified class names (FQCNs) are available:

- `com.simba.cassandra.jdbc4.Driver`
- `com.simba.cassandra.jdbc4.DataSource`

Note:

Support for JDBC 4.0 is deprecated, and will be removed in a future release of this driver. For more information, see the release notes.

To support JDBC 4.1, classes with the following FQCNs are available:

- `com.simba.cassandra.jdbc41.Driver`
- `com.simba.cassandra.jdbc41.DataSource`

To support JDBC 4.2, classes with the following FQCNs are available:

- `com.simba.cassandra.jdbc42.Driver`
- `com.simba.cassandra.jdbc42.DataSource`

The following sample code shows how to use the `DriverManager` class to establish a connection for JDBC 4.2:

```
private static Connection connectViaDM() throws Exception
{
   Connection connection = null;
   connection = DriverManager.getConnection(CONNECTION_URL);
   return connection;
}
```

The following sample code shows how to use the `DataSource` class to establish a connection:

```
private static Connection connectViaDS() throws Exception
{
   Connection connection = null;
   DataSource ds = new com.simba.cassandra.jdbc42.DataSource
   ();
   ds.setURL(CONNECTION_URL);
   connection = ds.getConnection();
   return connection;
}
```

Building the Connection URL

Use the connection URL to supply connection information to the database that you are accessing. Depending on whether you are connecting to a database that is hosted on a Cassandra server or an Astra instance, different connection properties are required.

- Cassandra Connection URLs on page 12
- Astra Connection URLs on page 13

! Important:

- Properties are case-sensitive.
- Do not duplicate properties in the connection URL.

Cassandra Connection URLs

When connecting to Cassandra, at minimum you must specify the DNS or IP address of the Cassandra server. Depending on the server configuration, other settings such as those related to authentication and SSL encryption might also be required.

The following is the format of a basic connection URL for connecting to Cassandra, where *[Host]* is the DNS or IP address of the server:

```
jdbc:cassandra://[Host]
```

By default, the driver connects to port 9042.

You can specify optional settings such as the number of the TCP port to connect to or any of the connection properties supported by the driver. For a list of the properties available in the driver, see Driver Configuration Options on page 31.

The following is the format of a Cassandra connection URL that specifies some optional settings:

```
jdbc:cassandra://[Host]:[Port];[Property1]=[Value];
[Property2]=[Value];...
```

For example, to connect to port 9000 on a Cassandra server installed on the local machine, and authenticate the connection using a user name and password, you would use the following connection URL:

```
jdbc:cassandra://localhost:9000;AuthMech=1;UID=simba;
PWD=simba123
```

Astra Connection URLs

When connecting to Astra, at minimum you must provide credentials for authentication and SSL encryption.

Note:

For more information about configuring authentication and SSL, see the following:

- Configuring Authentication on page 15
- Configuring SSL Connections on page 17

The following is the format of a basic connection URL for connecting to Astra:

```
jdbc:cassandra://;AuthMech=2;UID=[UserName];PWD=
[Password];SecureConnectionBundlePath="[BundlePath]"
```

The variables are defined as follows:

- *[UserName]* is the user name that you use to access the Astra database.
- *[Password]* is the password corresponding to your user name.

- *[BundlePath]* is the full path and name of the secure connection bundle associated with your Astra database. To make sure that the connection URL is compatible with all JDBC applications, escape the backslashes (\) in your file path by typing another backslash.

For example:

```
jdbc:cassandra://;AuthMech=2;UID=simba;PWD=simba123;SecureCo
nnectionBundlePath="C:\\Users\\admin\\Documents\\Astra\\secu
re-connect-simba_database.zip"
```

You can also specify optional settings, such as SSL settings for overriding the SSL certificates in the secure connection bundle. For a list of the properties available in the driver, see Driver Configuration Options on page 31.

The following is the format of an Astra connection URL that specifies some optional settings:

```
jdbc:cassandra://;AuthMech=2;UID=simba;PWD=simba123;SecureCo
nnectionBundlePath="C:\\Users\\admin\\Documents\\Astra\\secu
re-connect-simba_
database.zip";SSLMode=2;SSLTruststorePath="C:\\Users\\admin\
\Documents\\Astra\\astra_
trust.jks";SSLTrustStorePwd=jsmith123;SSLKeystorePath="C:\\U
sers\\admin\\Documents\\Astra\\astra_
keys.jks";SSLKeystorePwd=jsmithkeys123
```

Configuring Authentication

Some Cassandra databases require authentication for access, while all Astra databases require authentication. You can configure the Simba Cassandra JDBC Driver to authenticate your connection to the database by providing the necessary credentials.For more information, see the following:

- Configuring Authentication for Cassandra Databases on page 15
- Configuring Authentication for Astra Databases on page 15

You provide this information to the driver in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 12.

Configuring Authentication for Cassandra Databases

Some Cassandra databases require you to authenticate your connection by providing a user name and password.

To configure authentication for Cassandra databases:

1. Set the `AuthMech` property to `1`.
2. Set the `UID` property to an appropriate user name for accessing the database.
3. Set the `PWD` property to the password corresponding to the user name you provided.

For example:

```
jdbc:cassandra://localhost:9000;AuthMech=1;UID=simba;PWD=sim
ba123
```

Configuring Authentication for Astra Databases

To connect to an Astra database, you must authenticate your connection by providing a user name, password, and secure connection bundle.

To obtain the secure connection bundle associated with your Astra database, download it from the DataStax Constellation console. For more information, see "Obtaining database credentials" in the DataStax Astra documentation: https://docs.datastax.com/en/astra/aws/doc/dscloud/astra/dscloudObtainingCredentials.html.

To configure authentication for Astra databases:

1. Set the `AuthMech` property to `2`.
2. Set the `UID` property to an appropriate user name for accessing the database.
3. Set the `PWD` property to the password corresponding to the user name you provided.
4. Set the `SecureConnectionBundlePath` property to the full path and name of the secure connection bundle associated with your Astra database. To make sure that the connection URL is compatible with all JDBC applications, escape the backslashes (`\`) in your file path by typing another backslash.

For example:

```
jdbc:cassandra://;AuthMech=2;UID=simba;PWD=simba123;SecureCo
nnectionBundlePath="C:\\Users\\admin\\Documents\\Astra\\secu
re-connect-simba_database.zip"
```

Configuring SSL Connections

Note:

In this documentation, "SSL" indicates both TLS (Transport Layer Security) and SSL (Secure Sockets Layer). The driver supports industry-standard versions of TLS/SSL.

If you are connecting to a Cassandra server that has SSL enabled, you can configure the driver to connect to an SSL-enabled socket. When connecting to a server over SSL, the driver supports identity verification between the client (the driver itself) and the server.

If you are connecting to an Astra instance instead, then SSL encryption with identity verification between the client and the server is always required. Typically, the secure connection bundle that you use to authenticate your connection to Astra already includes the required SSL certificates, and the driver defaults to using those certificates when connecting to Astra, so you do not need to configure any additional SSL settings. However, you have the option of overriding the SSL certificates in the bundle by specifying other certificate information in your connection URL. For more information, see Configuring Two-Way SSL Authentication on page 18.

Configuring an SSL Connection without Identity Verification

You can configure a connection that uses SSL but does not verify the identity of the server or the driver.

You provide this information to the driver in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 12.

To configure an SSL connection without verification:

1. Set the `SSLMode` property to `1`.
2. Set the `UseSslIdentityCheck` property to `0`.

Configuring One-Way SSL Authentication

You can configure one-way SSL authentication so that the driver verifies the identity of the Cassandra server.

One-way authentication requires a TrustStore containing a signed, trusted SSL certificate for verifying the identity of the server. You can create a TrustStore and configure the driver to use it. If you do not configure the driver to use a specific TrustStore, then the driver uses the TrustStore named `jssecacerts`. If `jssecacerts` is not available, then the driver uses `cacerts` instead.

You provide this information to the driver in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 12.

To configure one-way SSL authentication:

1. If you are not using one of the default Java TrustStores, then configure the driver to access your TrustStore:
   a. Set the `SSLTruststorePath` property to the full path of the TrustStore.
   b. Set the `SSLTruststorePwd` property to your password for accessing the TrustStore.
2. Set the `SSLMode` property to `1`.
3. Set the `UseSslIdentityCheck` property to `1`.

Configuring Two-Way SSL Authentication

When connecting to Cassandra, you can configure two-way SSL verification so that the driver and the server verify each other.

Two-way authentication requires a TrustStore containing a signed, trusted SSL certificate for verifying the identity of the server, and a KeyStore containing a similar certificate for verifying the identity of the driver. You can create a TrustStore and configure the driver to use it. If you do not configure the driver to use a specific TrustStore, then the driver uses the TrustStore named `jssecacerts`. If `jssecacerts` is not available, then the driver uses `cacerts` instead.

When connecting to Astra, two-way SSL verification is always enabled, and the required certificates are typically provided through the secure connection bundle. However, you can override the SSL certificates in the bundle by configuring the driver to use a TrustStore and KeyStore instead.

These settings are configured in the connection URL. For more information about the syntax of the connection URL, see Building the Connection URL on page 12.

To configure two-way SSL verification:

1. If you are not using one of the default Java TrustStores, then configure the driver to access your TrustStore:

   a. Set the `SSLTruststorePath` property to the full path of the TrustStore.
   b. Set the `SSLTruststorePwd` property to your password for accessing the TrustStore.
2. Configure the driver to access your KeyStore:
   a. Set the `SSLKeystorePath` property to the full path of the KeyStore.
   b. Set the `SSLKeystorePwd` property to your password for accessing the KeyStore.
3. Set the `SSLMode` property to `2`.
4. If you are connecting to a Cassandra server, set the `UseSslIdentityCheck` property to `1`.

Configuring Logging

To help troubleshoot issues, you can enable logging in the driver.

! Important:

Only enable logging long enough to capture an issue. Logging decreases performance and can consume a large quantity of disk space.

The settings for logging apply to every connection that uses the Simba Cassandra JDBC Driver, so make sure to disable the feature after you are done using it.

In the connection URL, set the `LogLevel` key to enable logging at the desired level of detail. The following table lists the logging levels provided by the Simba Cassandra JDBC Driver, in order from least verbose to most verbose.

LogLevel Value	Description
`0`	Disable all logging.
`1`	Log severe error events that lead the driver to abort.
`2`	Log error events that might allow the driver to continue running.
`3`	Log events that might result in an error if action is not taken.
`4`	Log general information that describes the progress of the driver.
`5`	Log detailed information that is useful for debugging the driver.
`6`	Log all driver activity.

To enable logging:

1. Set the `LogLevel` property to the desired level of information to include in log files.
2. Set the `LogPath` property to the full path to the folder where you want to save log files. To make sure that the connection URL is compatible with all JDBC applications, escape the backslashes (`\`) in your file path by typing another backslash.

For example, the following connection URL enables logging level 3 and saves the log files in the `C:\temp` folder:

```
jdbc:cassandra://localhost;LogLevel=3;LogPath=C:\\temp
```

3. To make sure that the new settings take effect, restart your JDBC application and reconnect to the server.

The Simba Cassandra JDBC Driver produces the following log files in the location specified in the `LogPath` property:

- A `CassandraJDBC_driver.log` file that logs driver activity that is not specific to a connection.
- A `CassandraJDBC_connection_[Number].log` file for each connection made to the database, where *[Number]* is a number that identifies each log file. This file logs driver activity that is specific to the connection.

If the `LogPath` value is invalid, then the driver sends the logged information to the standard output stream (`System.out`).

To disable logging:

1. Set the `LogLevel` property to `0`.
2. To make sure that the new setting takes effect, restart your JDBC application and reconnect to the server.

Features

More information is provided on the following features of the Simba Cassandra JDBC Driver:

SQL Connector

The SQL Connector feature of the driver enables applications to execute standard SQL queries against Cassandra. It converts SQL-92 queries to CQL operations and processes the results. When the driver is configured to work in SQL With CQL Fallback mode, it uses the SQL Connector to handle SQL queries by loading and processing the data in memory. This feature enables the driver to support SQL operations that cannot be executed natively through CQL queries, such as column filtering and table joins.

Data Types

The Simba Cassandra JDBC Driver supports many common data formats, converting between CQL, SQL, and Java data types. The following table lists the supported data type mappings.

To support complex data types such as lists, maps, and sets, the driver renormalizes the data into virtual tables. For more information, see Virtual Tables on page 25.

Additionally, the driver can be configured to flatten UDT data into columns. For more information, see User-Defined Types on page 25.

CQL Type	SQL Type	Java Type
ASCII	VARCHAR	String

CQL Type	SQL Type	Java Type
BIGINT	BIGINT	Long
BLOB See the note below.	LONGVARBINARY	String
BOOLEAN	BOOLEAN	Long
COUNTER	BIGINT	Long
DATE	DATE	java.sql.Date
DECIMAL	DECIMAL See the note below.	java.math.BigDecimal
DOUBLE	DOUBLE	Double
FLOAT	REAL	Float
INET See the note below.	VARCHAR	String
INT	INTEGER	Integer
LIST	N/A The data is renormalized into a virtual table.	N/A
MAP	N/A The data is renormalized into a virtual table.	N/A
SET	N/A The data is renormalized into a virtual table.	N/A
SMALLINT	SMALLINT	Short

CQL Type	SQL Type	Java Type
TINYINT	TINYINT	Byte
TEXT	VARCHAR	String
TIME	TIME	java.sql.Time
TIMESTAMP See the note below.	TIMESTAMP	java.sql.Timestamp
TIMEUUID	VARCHAR	String
TUPLE	VARCHAR	String
UDT	VARCHAR	String
UUID	VARCHAR	String
VARCHAR	VARCHAR	String
VARINT	NUMERIC See the note below.	java.math.BigDecimal

Note:

- The scale and precision for data of type DECIMAL are determined by configurable settings in the driver. For more information, see DecimalColumnScale on page 32 and DecimalColumnPrecision on page 32.
- The precision for data of type NUMERIC are determined by configurable settings in the driver. For more information, see VarintColumnPrecision on page 45.
- If you insert a BLOB type, then the number of HEX digits must be even. 0xABC is not accepted as a valid binary literal. When you insert binary literal input as a string the driver truncates the last digit. "ABC" is inserted as "AB", for example.
- Cassandra internally represents a TIMESTAMP value as a 64-bit signed integer value representing the number of milliseconds since epoch January 1 1970 at 00:00:00 GMT. The range of TIMESTAMP values supported by the Simba Cassandra JDBC Driver is from "1601-01-01 00:00:00.000" to "9999-12-31 23:59:59.999".

User-Defined Types

The Simba Cassandra JDBC Driver can be configured to provide support for Cassandra user-defined types by flattening the underlying subtypes into their own individual columns in the table. Specifically, user-defined types are supported as base data or as part of a collection. For information about enabling this feature, see FlattenUDTColumn on page 36.

The column names for user-defined types are constructed in the following format:

[Column_name]_[Subtype_name]

Where:

- *[Column_name]* is the name of column that contains the user-defined type.
- *[Subtype_name]* is the name of the subtype in the user-defined type.

For example, a user-defined type that contains a user's full name is created as follows:

```
CREATE TYPE fullname
(first_name text, last_name text)
```

The driver creates a table that contains two columns, `id:text` and `name:fullname`. The driver then reads the data from this table as follows:

```
"id", "name_first_name", "name_last_name"
"a", "Chris", "Kwan"
```

Virtual Tables

One advantage of the Apache Cassandra design is the ability to store data that is denormalized into a fewer number of tables. By taking advantage of nested data structures such as sets, lists, and maps, transactions can be simplified. However, the JDBC interface does not natively support accessing this type of data. By renormalizing the data contained within collections (sets, lists, and maps) into virtual tables, the Simba Cassandra JDBC Driver allows users to directly interact with the data but leave the storage of the data in its denormalized form in Cassandra.

If a table contains any collection columns, when the table is queried for the first time, the driver creates the following virtual tables:

- A base table, which contains the same data as the real table except for the collection columns.
- A virtual table for each collection column, which expands the nested data.

Virtual tables refer to the data in the real table, enabling the driver to access the denormalized data. By querying the virtual tables, you can access the contents of Cassandra collections via JDBC. When you write or modify data in a virtual table, the data in the real table in the Cassandra database is updated.

The base table and virtual tables appear as additional tables in the list of tables that exist in the database. The base table uses the same name as the real table that it represents. The virtual tables that represent collections are named using the name of the real table, a separator (**_vt_** by default), and the name of the column.

For example, consider the following table. ExampleTable is a Cassandra database table that contains an integer primary key column named pk_int, a list column, a map column, and a set column (named StringSet).

pk_int	List	Map	StringSet
1	["1", "2" , "3"]	{ "S1" : "a", "S2" : "b" }	{ "A", "B", "C" }
3	["100", "101", "102", "105"]	{ "S1" : "t" }	{ "A", "E" }

The driver would generate multiple virtual tables to represent this single table. The first virtual table is the base table, which is shown below.

pk_int
1
3

The base table contains the same data as the original database table except for the collections, which are omitted from this table and expanded in other virtual tables.

The following three tables show the virtual tables that renormalize the data from the List, Map, and StringSet columns.

pk_int	List#index	List#value
1	0	1
1	1	2
1	2	3

pk_int	List#index	List#value
3	0	100
3	1	101
3	2	102
3	3	105

pk_int	Map#key	Map#value
1	S1	a
1	S2	b
3	S1	t

pk_int	StringSet#value
1	A
1	B
1	C
3	A
3	E

The foreign key columns in the virtual tables reference the primary key columns in the real table, and indicate which real table row the virtual table row corresponds to. The columns with names that end with #index or #key indicate the position of the data within the original list or map. The columns with names that end with #value contain the expanded data from the collection.

The data in the virtual tables can be selected, inserted, and updated as if they were normal tables, and the driver will handle the storage details within Cassandra. You can also explicitly append data to the end of a list by inserting a row of data with the index column set to -1.

For example, to append 106 to the List column in ExampleTable, where pk_int = 3, use the following query:

```
INSERT INTO "ExampleTable_vt_List" (pk_int, "List#index",
"List#value") VALUES (3, -1, '106')
```

Write-Back

The Simba Cassandra JDBC Driver supports Data Manipulation Languages (DML) statements such as INSERT, UPDATE, and DELETE.

Collection (LIST/MAP/SET) within collection DML operations are not supported.

Because Cassandra supports the UPSERT operation instead of INSERT and UPDATE, when you execute an INSERT or UPDATE statement using the Simba Cassandra JDBC Driver, the resulting behavior is an UPSERT operation. When you use the driver to write data to a Cassandra database, the INSERT and UPDATE operations both set the column value regardless of whether the data already exists.

You can use the TRUNCATE TABLE statement to delete rows from non-virtual tables. However, to delete rows from virtual tables, you must use the DELETE FROM statement instead.

Query Modes

The Simba Cassandra JDBC Driver can be configured to process queries as SQL statements or as CQL statements.

The default query mode used by the driver is SQL With CQL Fallback. In this query mode, the driver treats all incoming queries as SQL. If an error occurs while handling the query as SQL, then the driver will pass the original query to Cassandra to execute as CQL. However, because Cassandra is not aware of virtual tables, incoming queries that reference virtual tables will fail when they are passed through to the server to be executed as CQL.

You can configure the driver to work in CQL mode by setting the `QueryMode` connection property to `1`. In this query mode, the driver treats all incoming queries as CQL, so any queries written in a non-CQL syntax will fail.

Alternatively, you can configure the driver to work in SQL mode by setting the `QueryMode` connection property to `2`. When working in SQL mode, the driver treats all incoming queries as SQL, so any queries that are not written in standard SQL-92 syntax will fail.

When working in CQL mode, the driver does not expose any virtual tables. This happens because virtual table metadata is exposed only by the driver; Cassandra metadata does not contain any information relating to virtual tables. Virtual tables are only accessible via SQL mode.

Catalog and Schema Support

The Simba Cassandra JDBC Driver supports both catalogs and schemas to make it easy for the driver to work with various JDBC applications. Since Cassandra only organizes column families into keyspaces, the driver provides a synthetic catalog named CASSANDRA under which all of the keyspaces are organized. The driver also maps the JDBC schema to the Cassandra keyspace. The driver provides access to all of the schemas/databases that are listed under this catalog, ensuring compatibility with standard BI tools.

Security and Authentication

To protect data from unauthorized access, some Cassandra data stores require connections to be authenticated with user credentials or the SSL protocol. The Simba Cassandra JDBC Driver provides full support for these authentication protocols.

Note:

In this documentation, "SSL" indicates both TLS (Transport Layer Security) and SSL (Secure Sockets Layer). The driver supports industry-standard versions of TLS/SSL.

The driver provides a mechanism that allows you to authenticate your connection using your Cassandra user name and password. The driver also supports authentication to Astra databases, which require a user name, password, and secure connection bundle. For more information about configuring authentication for your connection, see Configuring Authentication on page 15.

Additionally, the driver supports the following types of SSL connections:

- No identity verification
- One-way authentication
- Two-way authentication

Depending on the configuration of your Cassandra server, you might have the option of connecting without using SSL encryption. However, Astra instances always require SSL encryption with two-way authentication.

It is recommended that you enable SSL whenever you connect to a server that is configured to support it. SSL encryption protects data and credentials when they are transferred over the network, and provides stronger security than authentication alone. For detailed configuration instructions, see Configuring SSL Connections on page 17.

The SSL version that the driver supports depends on the JVM version that you are using. For information about the SSL versions that are supported by each version of Java, see "Diagnosing TLS, SSL, and HTTPS" on the Java Platform Group Product Management Blog: https://blogs.oracle.com/java-platform-group/entry/diagnosing_tls_ssl_and_https.

Note:

The SSL version used for the connection is the highest version that is supported by both the driver and the server, which is determined at connection time.

Driver Configuration Options

Driver Configuration Options lists and describes the properties that you can use to configure the behavior of the Simba Cassandra JDBC Driver.

You can set configuration properties using the connection URL. For more information, see Building the Connection URL on page 12.

Note:

Property names and values are case-sensitive.

AuthMech

Default Value	Data Type	Required
`0`	Integer	No

Description

This property specifies whether the driver connects to a Cassandra or Astra database, and whether the driver authenticates the connection.

- `0`: The driver connects to a Cassandra database without authenticating the connection.
- `1`: The driver connects to a Cassandra database, and authenticates the connection using a user name and password.
- `2`: The driver connects to an Astra database, and authenticates the connection using a user name, password, and secure connection bundle.

BinaryColumnLength

Default Value	Data Type	Required
`4000`	Integer	No

Description

The maximum data length for BLOB columns.

ConnectTimeoutMillis

Default Value	Data Type	Required
The `DEFAULT_CONNECT_TIMEOUT_MILLIS` value configured on the server, which defaults to `5000`.	Integer	No

Description

The number of milliseconds that the driver waits before ending an attempt to connect to the server.

DecimalColumnPrecision

Default Value	Data Type	Required
`38`	Integer	No

Description

The precision that the driver uses when working with DECIMAL data.

Note:

The maximum value that you can set for this property is 32767.

DecimalColumnScale

Default Value	Data Type	Required
`10`	Integer	No

Description

The scale that the driver uses when working with DECIMAL data.

! Important:

- When you insert a value with a greater scale than this setting, the driver truncates the inserted value without returning any messages about the truncation.
- The maximum value that you can set for this property is 32767.

DefaultKeyspace

Default Value	Data Type	Required
None	String	No

Description

The default keyspace (schema) to connect to in Cassandra.

EnableAsynchronousWrites

Default Value	Data Type	Required
`1`	Integer	No

Description

This property specifies whether the driver uses asynchronous inserts or batch insertion.

- `1`: The driver uses asynchronous insertion.
- `0`: The driver uses batch insertion.

EnableCaseSensitive

Default Value	Data Type	Required
`1`	Integer	No

Description

This property specifies whether the driver differentiates between capital and lower-case letters in schema, table, and column names.

- `1`: The driver differentiates between capital and lower-case letters in schema, table, and column names. It is recommended that you enclose the names of all schemas, tables, and columns in double quotation marks (`"`) if this option is enabled.
- `0`: The driver ignores the capitalization of schema, table, and column names.

! Important:

- If the naming conventions for your Cassandra server are case-sensitive, you must leave this property set to `1`.
- If you are using the driver in a BI tool such as Tableau or Lumira, it is recommended that you leave this property set to `1`.
- If this property is set to `0`, then queries that use case-sensitive schema, table, and column names are not supported.

EnableLatencyAware

Default Value	Data Type	Required
`0`	Integer	No

Description

This property specifies whether the driver uses a latency-awareness algorithm to distribute the load away from slower-performing nodes.

- `1`: The driver uses the latency-awareness algorithm.
- `0`: The driver does not use the latency-awareness algorithm.

EnableNullInsert

Default Value	Data Type	Required
`0`	Integer	No

Description

This property specifies how the driver inserts NULL values.

- `1`: The driver inserts all NULL values as specified in INSERT statements.
- `0`: If an INSERT statement only specifies NULL values for a column or does not specify any values for a column, then the driver omits that column when executing the INSERT statement.

Consider the following before modifying this property:

- It is recommended that you leave this property set to the default value of `0` so that the driver does not insert NULL values into empty cells and create tombstones, which might decrease server performance and cause errors to occur. However, this setting might decrease driver performance when executing INSERT statements that affect a large number of rows.
- It is recommended that you set this property to `1` only when executing INSERT statements that do not contain unnecessary NULL values, because inserting NULL values into empty columns creates tombstones.

For more information about tombstones, see "About deletes" in the Apache Cassandra 2.0 documentation: http://docs.datastax.com/en/cassandra/2.0/cassandra/dml/dml_about_deletes_c.html.

EnablePaging

Default Value	Data Type	Required
`1`	Integer	No

Description

This property specifies whether to split large result sets into pages.

- `1`: The driver splits large result sets into pages.
- `0`: The driver fetches all results into memory regardless of the result set size.

Also, see the related driver configuration option RowsPerPage on page 40.

EnableTokenAware

Default Value	Data Type	Required
`1`	Integer	No

Description

This property specifies whether to use a token-aware policy to improve load balancing and latency.

- `1`: The driver uses the token-aware policy.
- `0`: The token-aware policy is not used.

FlattenUDTColumn

Default Value	Data Type	Required
`0`	Integer	No

Description

This property specifies how the driver handles user-defined types (UDT).

- `1`: The driver flattens UDT data into columns. During read and write operations, the subtypes and metadata of the UDT data are retained in the column typing and metadata.

  When this property is enabled, the following limitations also apply:

  - The driver cannot insert nested collections as VARCHAR data.
  - If a map contains two user-defined types with identical field names (that is, if the key and value of the map are of the same user-defined type), then a column name collision occurs.
  - If a table uses a user-defined type as its primary key, then when you execute an INSERT statement on that table, the fields of the primary key cannnot be NULL.
- `0`: The driver exposes UDT values as string values. The subtypes and metadata of the UDT data are not maintained during read and write operations, and the driver provides only partial support for write operations.

LoadBalancingPolicy

Default Value	Data Type	Required
`0`	Integer	No

Description

This property specifies the load balancing policy to be used.

- `1`: The Round Robin policy is used. The driver cycles through all nodes in the cluster on a per-query basis.
- `0`: The DC Aware policy is used. For each query, all nodes in a primary "local" data center are tried before any nodes from other data centers.

LogLevel

Default Value	Data Type	Required
`0`	Integer	No

Description

Use this property to enable or disable logging in the driver and to specify the amount of detail included in log files.

! Important:

Only enable logging long enough to capture an issue. Logging decreases performance and can consume a large quantity of disk space.

The settings for logging apply to every connection that uses the Simba Cassandra JDBC Driver, so make sure to disable the feature after you are done using it.

Set the property to one of the following numbers:

- `0`: Disable all logging.
- `1`: Enable logging on the FATAL level, which logs very severe error events that will lead the driver to abort.
- `2`: Enable logging on the ERROR level, which logs error events that might still allow the driver to continue running.

- `3`: Enable logging on the WARNING level, which logs events that might result in an error if action is not taken.
- `4`: Enable logging on the INFO level, which logs general information that describes the progress of the driver.
- `5`: Enable logging on the DEBUG level, which logs detailed information that is useful for debugging the driver.
- `6`: Enable logging on the TRACE level, which logs all driver activity.

When logging is enabled, the driver produces the following log files in the location specified in the `LogPath` property:

- A `CassandraJDBC_driver.log` file that logs driver activity that is not specific to a connection.
- A `CassandraJDBC_connection_`*`[Number]`*`.log` file for each connection made to the database, where *[Number]* is a number that identifies each log file. This file logs driver activity that is specific to the connection.

If the `LogPath` value is invalid, then the driver sends the logged information to the standard output stream (`System.out`).

LogPath

Default Value	Data Type	Required
The current working directory.	String	No

Description

The full path to the folder where the driver saves log files when logging is enabled.

Note:

To make sure that the connection URL is compatible with all JDBC applications, it is recommended that you escape the backslashes (`\`) in your file path by typing another backslash.

PWD

Default Value	Data Type	Required
None	String	Yes, if `AuthMech` is set to `1` or `2`.

Description

The password corresponding to the user name that you provided using the property UID on page 45.

QueryMode

Default Value	Data Type	Required
`0`	Integer	No

Description

This property specifies the query mode to use when sending queries to Cassandra.

- `0`: The driver uses the SQL With CQL Fallback query mode. The driver executes all queries in SQL by default, but if a query fails, then the driver executes the query in CQL.
- `1`: The driver uses the CQL query mode and executes all queries in CQL.
- `2`: The driver uses the SQL query mode and executes all queries in SQL.

Note:

When working in CQL mode, the driver does not expose any virtual tables. This happens because virtual table metadata is exposed only by the driver; Cassandra metadata does not contain any information relating to virtual tables. Virtual tables are only accessible via SQL mode.

ReadTimeoutMillis

Default Value	Data Type	Required
The `DEFAULT_READ_TIMEOUT_MILLIS` value configured on the server, which defaults to `12000`.	Integer	No

Description

The number of milliseconds that the driver waits before ending an attempt to read data from a particular server node.

If you set this property to `0`, then timeouts are disabled for read operations.

Note:

- Because this timeout behavior is applied on a per-node basis, if your query is executed on more than one node, the actual interval of time that passes before the query times out may differ from your `ReadTimeoutMillis` setting.
- It is recommended that you set this property to a value that is greater than the other timeout settings configured on the server. Timeout settings for the Cassandra server are determined by the properties in the `cassandra.yaml` file that are suffixed with `_request_timeout_in_ms`.

RowsPerPage

Default Value	Data Type	Required
`10000`	Integer	No

Description

When result set paging is enabled, use this option to specify the maximum number of rows to display on each page.

Also, see the related driver configuration option EnablePaging on page 35.

SecureConnectionBundlePath

Default Value	Data Type	Required
None	String	Yes, if connecting to an Astra database.

Description

The full path and name of the secure connection bundle associated with your Astra database.

Note:

To make sure that the connection URL is compatible with all JDBC applications, escape the backslashes (\) in your file path by typing another backslash

You can download the secure connection bundle from the DataStax Constellation console. For more information, see "Obtaining database credentials" in the DataStax Astra documentation: https://docs.datastax.com/en/astra/aws/doc/dscloud/astra/dscloudObtainingCredentials.html.

SSLMode

Default Value	Data Type	Required
`0` when connecting to Cassandra, or `3` when connecting to Astra.	Integer	No

Description

This property specifies how the driver uses SSL to connect to the server.

If you are connecting to a Cassandra database, set this property to one of the following values:

- `0`: The driver does not use SSL to connect to the server.

- `1`: If the `UseSslIdentityCheck` property is also set to `1`, the driver connects using one-way SSL authentication. Otherwise, the driver connects to the server using SSL but does not verify the identity of the server.
- `2`: If the `UseSslIdentityCheck` property is also set to `1`, the driver connects using two-way SSL authentication. Otherwise, the driver connects to the server using SSL, but the driver and the server do not verify each other.

Or, if you are connecting to an Astra database, set this property to one of the following values:

- `2`: The driver connects using two-way SSL authentication, and overrides the SSL certificates in the secure connection bundle. For more information, see Configuring SSL Connections on page 17.
- `3`: The driver connects using two-way SSL authentication, and uses the SSL certificates that are included in the secure connection bundle. For more information, see SecureConnectionBundlePath on page 41.

SSLKeystorePath

Default Value	Data Type	Required
None	String	Yes, if `SSLMode=2`.

Description

The full path of the Java KeyStore containing the certificate for verifying the client.

See also the property SSLKeystorePwd on page 42.

SSLKeystorePwd

Default Value	Data Type	Required
None	String	No

Description

The password to use when checking the integrity of the KeyStore.

See also the property SSLKeystorePath on page 42.

SSLTruststorePath

Default Value	Data Type	Required
`jssecacerts`, if it exists. If `jssecacerts` does not exist, then `cacerts` is used. The default location of `cacerts` is `jre\lib\security\`.	String	Yes, if `SSLMode=1` or `SSLMode=2`.

Description

Note:

This property is applicable only when `SSLMode` is set to `1` or `2`. For more information, see Configuring SSL Connections on page 17.

The full path of the Java TrustStore containing an SSL certificate for verifying the server.

See also the property SSLTruststorePwd on page 43.

SSLTruststorePwd

Default Value	Data Type	Required
None	String	Yes, if `SSLMode=1` or `SSLMode=2`.

Description

Note:

This property is applicable only when `SSLMode` is set to `1` or `2`. For more information, see Configuring SSL Connections on page 17.

The password to use when checking the integrity of the TrustStore.

See also the property SSLTruststorePath on page 43.

StringColumnLength

Default Value	Data Type	Required
`4000`	Integer	No

Description

The maximum data length for ASCII, TEXT, and VARCHAR columns.

TunableConsistency

Default Value	Data Type	Required
`1`	Integer	No

Description

The specific Cassandra replica or the number of Cassandra replicas that must process a query in order for the query to be considered successful.

The following values are supported, where each value corresponds to one of the consistency levels available in Cassandra:

- `0` for ANY
- `1` for ONE
- `2` for TWO
- `3` for THREE
- `4` for QUORUM
- `5` for ALL
- `6` for LOCAL_QUORUM
- `7` for EACH_QUORUM
- `10` for LOCAL_ONE

For detailed information about each consistency level, see "Configuring data consistency" in the Apache Cassandra 2.0 documentation: http://www.datastax.com/documentation/cassandra/2.0/cassandra/dml/dml_config_consistency_c.html.

UID

Default Value	Data Type	Required
None	String	Yes, if `AuthMech` is set to `1` or `2`.

Description

The user name that you use to access the database.

UseSslIdentityCheck

Default Value	Data Type	Required
`1`	Integer	No

Description

Note:

This property is not applicable to Astra connections.

This property specifies whether the driver requires the host name of the server to match the host name in the certificate during SSL verification.

- `1`: The driver requires the host name of the server to match the host name in the certificate.
- `0`: The driver allows the host name of the server to not match the host name in the certificate.

VarintColumnPrecision

Default Value	Data Type	Required
`38`	Integer	No

Description

The precision that the driver uses when working with NUMERIC data.

Note:

The maximum value that you can set for this property is 32767.

VTTableNameSeparator

Default Value	Data Type	Required
`_vt_`	String	No

Description

The separator for naming a virtual table built from a collection.

The name of a virtual table consists of the name of the table in the database, then the separator, and then the name of the collection.

For example, `DatabaseTableName_vt_CollectionName`.

For more information about virtual tables, see Virtual Tables on page 25.

Third-Party Trademarks