



TIBCO® Data Virtualization

SparkSQL Adapter Guide

Version 8.7.0 | October 2023

Contents

Contents	2
SparkSQL Adapter	4
SparkSQL Version Support	4
SQL Compliance	4
Getting Started	4
Basic Tab	5
Logging	7
Using Kerberos	9
Changelog	10
Advanced Features	14
User Defined Views	15
SSL Configuration	17
Firewall and Proxy	18
Query Processing	18
Logging	19
SQL Compliance	22
SELECT Statements	23
SELECT INTO Statements	90
INSERT Statements	91
EXECUTE Statements	91
PIVOT and UNPIVOT	92
Data Model	93
Connection String Options	94
Authentication	98
Kerberos	104
SSL	109
Firewall	116

Proxy	119
Logging	126
Schema	127
Miscellaneous	128
SparkSQL Adapter Limitations	137
TIBCO Product Documentation and Support Services	138
How to Access TIBCO Documentation	138
How to Contact TIBCO Support	139
Release Version Support	139
How to Join TIBCO Community	140
Legal and Third-Party Notices	141

SparkSQL Adapter

SparkSQL Version Support

The adapter leverages Spark Thrift to enable bidirectional SQL access to SparkSQL data. Spark version 1.6 and above are supported.

SQL Compliance

The [SQL Compliance](#) section shows the SQL syntax supported by the adapter and points out any limitations.

Getting Started

Connecting to SparkSQL

[Basic Tab](#) shows how to authenticate to SparkSQL and configure any necessary connection properties. Additional adapter capabilities can be configured using the available [Connection](#) properties on the Advanced tab. The Advanced Settings section shows how to set up more advanced configurations and troubleshoot connection errors.

Deploying the SparkSQL Adapter

To deploy the adapter, you can execute the server_util utility via the command line by

1. Unzip the tdv.sparksql.zip file to the location of your choice.
2. Open a command prompt window.
3. Navigate to the <TDV_install_dir>/bin
4. Enter the server_util command with the -deploy option:

```
server_util -server <hostname> [-port <port>] -user <user> -
```

```
password <password> -deploy -package <TDV_install_dir>/adapters/tdv.sparksql/tdv.sparksql.jar
```

Note: When deploying a build of an existing adapter, you will need to undeploy the existing adapter using the `server_util` command with the `-undeploy` option.

```
server_util -server <hostname> [-port <port>] -user <user> -password <password> -undeploy -version 1 -name SparkSQL
```

Basic Tab

Connecting to SparkSQL

Specify the following to establish a connection with SparkSQL:

- Server: Set this to the host name or IP address of the server hosting SparkSQL.
- Port: Set this to the port for the connection to the SparkSQL instance.
- TransportMode: The transport mode to use to communicate with the SparkSQL server. Accepted entries are BINARY and HTTP. BINARY is selected by default.

Securing SparkSQL Connections

To enable TLS/SSL in the adapter, set UseSSL to True.

Authenticating to SparkSQL

The service may be authenticated to using the PLAIN, LDAP, NOSASL, KERBEROS auth schemes.

PLAIN

To authenticate with PLAIN, set the following connection properties:

- AuthScheme: Set this to PLAIN.
- User: Set this to user to login as.
- Password: Set this to the password of the user.

To authenticate, set User and Password.

LDAP

To authenticate with LDAP, set the following connection properties:

- AuthScheme: Set this to LDAP.
- User: Set this to user to login as.
- Password: Set this to the password of the user.

To authenticate, set User, Password, and AuthScheme.

NOSASL

When using NOSASL, no authentication is performed. Set the following connection properties:

- AuthScheme: Set this to NOSASL.

Kerberos

Please see [Using Kerberos](#) for details on how to authenticate with Kerberos.

Connecting to Databricks

To connect to a Databricks cluster, set the properties as described below. Note: The needed values can be found in your Databricks instance by navigating to 'Clusters', selecting the desired cluster, and selecting the JDBC/ODBC tab under 'Advanced Options'.

- Server: Set to the Server Hostname of your Databricks cluster.
- Port: 443

- TransportMode: HTTP
- HTTPPath: Set to the HTTP Path of your Databricks cluster.
- UseSSL: True
- AuthScheme: PLAIN
- User: Set this to user to login as
- Password: Set to your personal access token (value can be obtained by navigating to the User Settings page of your Databricks instance and selecting the Access Tokens tab).

Logging

The adapter uses TDV Server's logging (log4j) to generate log files. The settings within the TDV Server's logging (log4j) configuration file are used by the adapter to determine the type of messages to log. The following categories can be specified:

- Error: Only error messages are logged.
- Info: Both Error and Info messages are logged.
- Debug: Error, Info, and Debug messages are logged.

The Other property of the adapter can be used to set Verbosity to specify the amount of detail to be included in the log file, that is:

```
Verbosity=4;
```

You can use Verbosity to specify the amount of detail to include in the log within a category. The following verbosity levels are mapped to the log4j categories:

- 0 = Error
- 1-2 = Info
- 3-5 = Debug

For example, if the log4j category is set to DEBUG, the Verbosity option can be set to 3 for the minimum amount of debug information or 5 for the maximum amount of debug information.

Note that the log4j settings override the Verbosity level specified. The adapter never logs at a Verbosity level greater than what is configured in the log4j properties. In addition, if Verbosity is set to a level less than the log4j category configured, Verbosity defaults to the

minimum value for that particular category. For example, if Verbosity is set to a value less than 3 and the Debug category is specified, the Verbosity defaults to 3.

The following list is an explanation of the Verbosity levels and the information that they log.

- 1 - Will log the query, the number of rows returned by it, the start of execution and the time taken, and any errors.
- 2 - Will log everything included in Verbosity 1 and HTTP headers.
- 3 - Will additionally log the body of the HTTP requests.
- 4 - Will additionally log transport-level communication with the data source. This includes SSL negotiation.
- 5 - Will additionally log communication with the data source and additional details that may be helpful in troubleshooting problems. This includes interface commands.

Configure Logging for the SparkSQL Adapter

By default, logging is turned on without debugging. If debugging information is desired, uncomment the following line in the TDV Server's log4j.properties file (default location of this file is: C:\Program Files\TIBCO\TDV Server <version>\conf\server):

```
log4j.logger.com.cdata=DEBUG
```

The TDV Server must be restarted after changing the log4j.properties file, which can be accomplished by running the composite.bat script located at: C:\Program Files\TIBCO\TDV Server <version>\bin. Note that reauthenticating to the TDV Studio is required after restarting the server.

Here is an example of the calls:

```
.\composite.bat monitor restart
```

All logs for the adapter are written to the "cs_server_dsdc.log" file as specified in the log4j properties.

Note: The "log4j.logger.com.cdata=DEBUG" option is not required if the **Debug Output Enabled** option is set to true within the TDV Studio. To set this option, navigate to **Administrator > Configuration**. Select **Server > Configuration > Debugging** and set the Debug Output Enabled option to **True**.

Using Kerberos

This section shows how to use the adapter to authenticate using Kerberos.

Kerberos

To authenticate to SparkSQL using Kerberos, set the following properties:

- **KerberosKDC**: Set this to the host name or IP Address of your Kerberos KDC machine.
- **KerberosRealm**: Set this to **the realm of the SparkSQL Kerberos principal**. This will be the value after the '@' symbol (for instance, EXAMPLE.COM) of the **principal value** (for instance, ServiceName/MyHost@EXAMPLE.COM).
- **KerberosSPN**: Set this to the service and host of the SparkSQL Kerberos Principal. This is the value prior to the '@' symbol (for instance, ServiceName/MyHost) of the principal value (for instance, ServiceName/MyHost@EXAMPLE.COM).

Retrieve the Kerberos Ticket

You can use one of the following options to retrieve the required Kerberos ticket.

MIT Kerberos Credential Cache File

This option enables you to use the MIT Kerberos Ticket Manager or kinit command to get tickets. Note that you do not need to set the User or Password connection properties with this option.

1. Ensure that you have an environment variable created called **KRB5CCNAME**.
2. Set the **KRB5CCNAME** environment variable to a path pointing to your credential cache file (for instance, `C:\krb_cache\krb5cc_0` or `/tmp/krb5cc_0`). This file is created when generating your ticket with MIT Kerberos Ticket Manager.
3. To obtain a ticket, open the MIT Kerberos Ticket Manager application, click **Get Ticket**, enter your principal name and password, then click **OK**. If successful, ticket information appears in Kerberos Ticket Manager and is stored in the credential cache file.
4. Now that you have created the credential cache file, the adapter uses the cache file to obtain the Kerberos ticket to connect to SparkSQL.

As an alternative to setting the **KRB5CCNAME** environment variable, you can directly set the file path using the KerberosTicketCache property. When set, the adapter uses the specified cache file to obtain the Kerberos ticket to connect to SparkSQL.

Keytab File

If the **KRB5CCNAME environment variable has not been set**, you can retrieve a Kerberos ticket using a Keytab File. To do so, set the User property to the desired username and set the KerberosKeytabFile property to a file path pointing to the keytab file associated with the user.

User and Password

If both the **KRB5CCNAME** environment variable and the KerberosKeytabFile property have not been set, you can retrieve a ticket using a user and password combination. To do this, set the User and Password properties to the user/password combination that you use to authenticate with SparkSQL.

Cross-Realm

More complex Kerberos environments may require cross-realm authentication where multiple realms and KDC servers are used (e.g., where one realm/KDC is used for user authentication and another realm/KDC is used for obtaining the service ticket).

In such an environment, set the KerberosRealm and KerberosKDC properties to the values required for user authentication. Also set the KerberosServiceRealm and KerberosServiceKDC properties to the values required to obtain the service ticket.

Changelog

General Changes

Date	Build Number	Change Type	Description
------	--------------	-------------	-------------

12/14/2022	8383	General	Changed <ul style="list-style-type: none"> Added the Default column to the sys_procedureparameters table.
09/30/2022	8308	General	Changed <ul style="list-style-type: none"> Added the IsPath column to the sys_procedureparameters table.
08/17/2022	8264	General	Changed <ul style="list-style-type: none"> We now support handling the keyword "COLLATE" as standard function name as well.
12/03/2021	8007	SparkSQL	Deprecated <ul style="list-style-type: none"> The connection properties UseShowDatabasesQuery, UseDescTableQuery, and UseShowTablesQuery are deprecated. They were not being utilized by the driver.
11/15/2021	7989	SparkSQL	Added <ul style="list-style-type: none"> Support for TLS over SSH connections.
09/16/2021	7929	SparkSQL	Added <ul style="list-style-type: none"> Added four connection properties SSLClientCert, SSLClientCertPassword, SSLClientCertSubject, SSLClientCertType.
09/02/2021	7915	General	Added <ul style="list-style-type: none"> Added support for the STRING_SPLIT table-valued function in the CROSS APPLY clause.
08/07/2021	7889	General	Changed <ul style="list-style-type: none"> Added the KeySeq column to the sys_

foreignkeys table.			
08/06/2021	7888	General	Changed <ul style="list-style-type: none"> Added the new sys_primarykeys system table.
07/23/2021	7874	General	Changed <ul style="list-style-type: none"> Updated the Literal Function Names for relative date/datetime functions. Previously relative date/datetime functions resolved to a different value when used in the projection vs te predicate. Ie: SELECT LAST_MONTH() AS lm, Col FROM Table WHERE Col > LAST_MONTH(). Formerly the two LAST_MONTH() methods would resolve to different datetimes. Now they will match. As a replacement for the previous behavior, the relative date/datetime functions in the criteria may have an 'L' appended to them. Ie: WHERE col > L_LAST_MONTH(). This will continue to resolve to the same values that previously were calculated in the criteria. Note that the "L_" prefix will only work in the predicate - it not available for the projection.
07/08/2021	7859	General	Added <ul style="list-style-type: none"> Added the TCP Logging Module for the logging information happening on the TCP wire protocol. The transport bytes that are incoming and ongoing will be logged at verbosity=5.
05/31/2021	7821	SparkSQL	Added <ul style="list-style-type: none"> Added a new connection property

			SaslQop to support the "auth-conf" value in the "hive.server2.thrift.sasl.qop" property.
04/23/2021	7785	General	Added <ul style="list-style-type: none"> Added support for handling client side formulas during insert / update. For example: UPDATE Table SET Col1 = Concat(Col1, " - ", Col2) WHERE Col2 LIKE 'A%'
04/23/2021	7783	General	Changed <ul style="list-style-type: none"> Updated how display sizes are determined for varchar primary key and foreign key columns so they will match the reported length of the column.
04/16/2021	7776	General	Added <ul style="list-style-type: none"> Non-conditional updates between two columns is now available to all drivers. For example: UPDATE Table SET Col1=Col2 Changed <ul style="list-style-type: none"> Reduced the length to 255 for varchar primary key and foreign key columns. Updated implicit and metadata caching to improve performance and support for multiple connections. Old metadata caches are not compatible - you would need to generate new metadata caches if you are currently using CacheMetadata. Updated index naming convention to avoid duplicates Updated and standardized Getting Started connection help.

			<ul style="list-style-type: none"> Added the Advanced Features section to the help of all drivers. Categorized connection property listings in the help for all editions.
04/15 /2021	7775	General	Changed <ul style="list-style-type: none"> Kerberos authentication is updated to use TCP by default, but will fall back to UDP if a TCP connection cannot be established

Advanced Features

This section details a selection of advanced features of the SparkSQL adapter.

User Defined Views

The adapter allows you to define virtual tables, called *user defined views*, whose contents are decided by a pre-configured query. These views are useful when you cannot directly control queries being issued to the drivers. See [User Defined Views](#) for an overview of creating and configuring custom views.

SSL Configuration

Use [SSL Configuration](#) to adjust how adapter handles TLS/SSL certificate negotiations. You can choose from various certificate formats; see the [SSLServerCert](#) property under "Connection String Options" for more information.

Firewall and Proxy

Configure the adapter for compliance with [Firewall and Proxy](#), including Windows proxies and HTTP proxies. You can also set up tunnel connections.

Query Processing

The adapter offloads as much of the SELECT statement processing as possible to SparkSQL and then processes the rest of the query in memory (client-side).

See [Query Processing](#) for more information.

Logging

See [Logging](#) for an overview of configuration settings that can be used to refine CData logging. For basic logging, you only need to set two connection properties, but there are numerous features that support more refined logging, where you can select subsets of information to be logged using the [LogModules](#) connection property.

User Defined Views

The SparkSQL Adapter allows you to define a virtual table whose contents are decided by a pre-configured query. These are called *User Defined Views*, which are useful in situations where you cannot directly control the query being issued to the driver, e.g. when using the driver from a tool. The User Defined Views can be used to define predicates that are always applied. If you specify additional predicates in the query to the view, they are combined with the query already defined as part of the view.

There are two ways to create user defined views:

- Create a JSON-formatted configuration file defining the views you want.
- DDL statements.

Defining Views Using a Configuration File

User Defined Views are defined in a JSON-formatted configuration file called *UserDefinedViews.json*. The adapter automatically detects the views specified in this file.

You can also have multiple view definitions and control them using the [UserDefinedViews](#) connection property. When you use this property, only the specified views are seen by the adapter.

This User Defined View configuration file is formatted as follows:

- Each root element defines the name of a view.
- Each root element contains a child element, called **query**, which contains the custom

SQL query for the view.

For example:

```
{
  "MyView": {
    "query": "SELECT * FROM Customers WHERE MyColumn = 'value'"
  },
  "MyView2": {
    "query": "SELECT * FROM MyTable WHERE Id IN (1,2,3)"
  }
}
```

Use the UserDefinedViews connection property to specify the location of your JSON configuration file. For example:

```
"UserDefinedViews",
"C:\\Users\\yourusername\\Desktop\\tmp\\UserDefinedViews.json"
```

Defining Views Using DDL Statements

The adapter is also capable of creating and altering the schema via DDL Statements such as CREATE LOCAL VIEW, ALTER LOCAL VIEW, and DROP LOCAL VIEW.

Create a View

To create a new view using DDL statements, provide the view name and query as follows:

```
CREATE LOCAL VIEW [MyViewName] AS SELECT * FROM Customers LIMIT 20;
```

If no JSON file exists, the above code creates one. The view is then created in the JSON configuration file and is now discoverable. The JSON file location is specified by the UserDefinedViews connection property.

Alter a View

To alter an existing view, provide the name of an existing view alongside the new query you would like to use instead:


```
ALTER LOCAL VIEW [MyViewName] AS SELECT * FROM Customers WHERE
TimeModified > '3/1/2020';
```

The view is then updated in the JSON configuration file.

Drop a View

To drop an existing view, provide the name of an existing schema alongside the new query you would like to use instead.

```
DROP LOCAL VIEW [MyViewName]
```

This removes the view from the JSON configuration file. It can no longer be queried.

Schema for User Defined Views

User Defined Views are exposed in the **UserViews** schema by default. This is done to avoid the view's name clashing with an actual entity in the data model. You can change the name of the schema used for UserViews by setting the UserViewsSchemaName property.

Working with User Defined Views

For example, a SQL statement with a User Defined View called *UserViews.RCustomers* only lists customers in Raleigh:

```
SELECT * FROM Customers WHERE City = 'Raleigh';
```

An example of a query to the driver:

```
SELECT * FROM UserViews.RCustomers WHERE Status = 'Active';
```

Resulting in the effective query to the source:

```
SELECT * FROM Customers WHERE City = 'Raleigh' AND Status = 'Active';
```

That is a very simple example of a query to a User Defined View that is effectively a combination of the view query and the view definition. It is possible to compose these queries in much more complex patterns. All SQL operations are allowed in both queries and are combined when appropriate.

SSL Configuration

Customizing the SSL Configuration

By default, the adapter attempts to negotiate SSL/TLS by checking the server's certificate against the system's trusted certificate store.

To specify another certificate, see the [SSLServerCert](#) property for the available formats to do so.

Firewall and Proxy

Connecting Through a Firewall or Proxy

HTTP Proxies

To connect through the Windows system proxy, you do not need to set any additional connection properties. To connect to other proxies, set [ProxyAutoDetect](#) to false.

In addition, to authenticate to an HTTP proxy, set [ProxyAuthScheme](#), [ProxyUser](#), and [ProxyPassword](#), in addition to [ProxyServer](#) and [ProxyPort](#).

Other Proxies

Set the following properties:

- To use a proxy-based firewall, set [FirewallType](#), [FirewallServer](#), and [FirewallPort](#).
- To tunnel the connection, set [FirewallType](#) to TUNNEL.
- To authenticate, specify [FirewallUser](#) and [FirewallPassword](#).
- To authenticate to a SOCKS proxy, additionally set [FirewallType](#) to SOCKS5.

Query Processing

Query Processing

CData has a client-side SQL engine built into the adapter library. This enables support for

the full capabilities that SQL-92 offers, including filters, aggregations, functions, etc.

For sources that do not support SQL-92, the adapter offloads as much of SQL statement processing as possible to SparkSQL and then processes the rest of the query in memory (client-side). This results in optimal performance.

For data sources with limited query capabilities, the adapter handles transformations of the SQL query to make it simpler for the adapter. The goal is to make smart decisions based on the query capabilities of the data source to push down as much of the computation as possible. The SparkSQL Query Evaluation component examines SQL queries and returns information indicating what parts of the query the adapter is not capable of executing natively.

The SparkSQL Query Slicer component is used in more specific cases to separate a single query into multiple independent queries. The client-side Query Engine makes decisions about simplifying queries, breaking queries into multiple queries, and pushing down or computing aggregations on the client-side while minimizing the size of the result set.

There's a significant trade-off in evaluating queries, even partially, client-side. There are always queries that are impossible to execute efficiently in this model, and some can be particularly expensive to compute in this manner. CData always pushes down as much of the query as is feasible for the data source to generate the most efficient query possible and provide the most flexible query capabilities.

More Information

For a full discussion of how CData handles query processing, see [CData Architecture: Query Execution](#).

Logging

Capturing adapter logging can be very helpful when diagnosing error messages or other unexpected behavior.

Basic Logging

You will simply need to set two connection properties to begin capturing adapter logging.

- **Logfile:** A filepath which designates the name and location of the log file.
- **Verbosity:** This is a numerical value (1-5) that determines the amount of detail in the log. See the page in the Connection Properties section for an explanation of the five

levels.

- MaxLogFileSize: When the limit is hit, a new log is created in the same folder with the date and time appended to the end. The default limit is 100 MB. Values lower than 100 kB will use 100 kB as the value instead.
- MaxLogFileCount: A string specifying the maximum file count of log files. When the limit is hit, a new log is created in the same folder with the date and time appended to the end and the oldest log file will be deleted. Minimum supported value is 2. A value of 0 or a negative value indicates no limit on the count.

Once this property is set, the adapter will populate the log file as it carries out various tasks, such as when authentication is performed or queries are executed. If the specified file doesn't already exist, it will be created.

Log Verbosity

The verbosity level determines the amount of detail that the adapter reports to the Logfile. Verbosity levels from 1 to 5 are supported. These are described in the following list:

1	Setting <u>Verbosity</u> to 1 will log the query, the number of rows returned by it, the start of execution and the time taken, and any errors.
2	Setting <u>Verbosity</u> to 2 will log everything included in <u>Verbosity</u> 1 and additional information about the request.
3	Setting <u>Verbosity</u> to 3 will additionally log HTTP headers, as well as the body of the request and the response.
4	Setting <u>Verbosity</u> to 4 will additionally log transport-level communication with the data source. This includes SSL negotiation.
5	Setting <u>Verbosity</u> to 5 will additionally log communication with the data source and additional details that may be helpful in troubleshooting problems. This includes interface commands.

The Verbosity should not be set to greater than 1 for normal operation. Substantial amounts of data can be logged at higher verbosities, which can delay execution times.

To refine the logged content further by showing/hiding specific categories of information, see [LogModules](#).

Sensitive Data

Verbosity levels 3 and higher may capture information that you do not want shared outside of your organization. The following lists information of concern for each level:

- Verbosity 3: The full body of the request and the response, which includes all the data returned by the adapter
- Verbosity 4: SSL certificates
- Verbosity 5: Any extra transfer data not included at Verbosity 3, such as non human-readable binary transfer data

Best Practices for Data Security

Although we mask sensitive values, such as passwords, in the connection string and any request in the log, it is always best practice to review the logs for any sensitive information before sharing outside your organization.

Java Logging

When Java logging is enabled in [Logfile](#), the [Verbosity](#) will instead map to the following logging levels.

- 0: Level.WARNING
- 1: Level.INFO
- 2: Level.CONFIG
- 3: Level.FINE
- 4: Level.FINER
- 5: Level.FINEST

Advanced Logging

You may want to refine the exact information that is recorded to the log file. This can be accomplished using the [LogModules](#) property.

This property allows you to filter the logging using a semicolon-separated list of logging modules.

All modules are four characters long. **Please note that modules containing three letters have a required trailing blank space.** The available modules are:

- **EXEC:** Query Execution. Includes execution messages for original SQL queries, parsed SQL queries, and normalized SQL queries. Query and page success/failure messages appear here as well.
- **INFO:** General Information. Includes the connection string, driver version (build number), and initial connection messages.
- **HTTP:** HTTP Protocol messages. Includes HTTP requests/responses (including POST messages), as well as Kerberos related messages.
- **SSL :** SSL certificate messages.
- **OAUT:** OAuth related failure/success messages.
- **SQL :** Includes SQL transactions, SQL bulk transfer messages, and SQL result set messages.
- **META:** Metadata cache and schema messages.
- **TCP :** Incoming and Ongoing raw bytes on TCP transport layer messages.

An example value for this property would be.

```
LogModules=INFO;EXEC;SSL ;SQL ;META;
```

Note that these modules refine the information as it is pulled after taking the Verbosity into account.

SQL Compliance

The SparkSQL Adapter supports several operations on data, including querying, deleting, modifying, and inserting.

SELECT Statements

See [SELECT Statements](#) for a syntax reference and examples.

See [Data Model](#) for information on the capabilities of the SparkSQL API.

INSERT Statements

See [INSERT Statements](#) for a syntax reference and examples.

EXECUTE Statements

Use EXECUTE or EXEC statements to execute stored procedures. See [EXECUTE Statements](#) for a syntax reference and examples.

Names and Quoting

- Table and column names are considered identifier names; as such, they are restricted to the following characters: [A-Z, a-z, 0-9, _:@].
- To use a table or column name with characters not listed above, the name must be quoted using double quotes ("name") in any SQL statement.
- Strings must be quoted using single quotes (e.g., 'John Doe').

SELECT Statements

A SELECT statement can consist of the following basic clauses.

- SELECT
- INTO
- FROM
- JOIN
- WHERE
- GROUP BY
- HAVING
- UNION
- ORDER BY
- LIMIT

SELECT Syntax

The following syntax diagram outlines the syntax supported by the SparkSQL adapter:

```

SELECT {
  [ TOP <numeric_literal> | DISTINCT ]
  {
    *
    | {
        <expression> [ [ AS ] <column_reference> ]
        | { <table_name> | <correlation_name> } .*
      } [ , ... ]
    }
  [ INTO csv:// [ filename= ] <file_path> [ ;delimiter=tab ] ]
  {
    FROM <table_reference> [ [ AS ] <identifier> ]
  } [ , ... ]
  [ [
    INNER | { { LEFT | RIGHT | FULL } [ OUTER ] }
    ] JOIN <table_reference> [ ON <search_condition> ] [ [ AS ]
<identifier> ]
  ] [ ... ]
  [ WHERE <search_condition> ]
  [ GROUP BY <column_reference> [ , ... ]
  [ HAVING <search_condition> ]
  [
    ORDER BY
    <column_reference> [ ASC | DESC ] [ NULLS FIRST | NULLS LAST ]
  ]
  [
    LIMIT <expression>
  ]
} | SCOPE_IDENTITY()
<expression> ::=
  | <column_reference>
  | @ <parameter>
  | ?
  | COUNT( * | { [ DISTINCT ] <expression> } )
  | { AVG | MAX | MIN | SUM | COUNT } ( <expression> )
  | NULLIF ( <expression> , <expression> )
  | COALESCE ( <expression> , ... )
  | CASE <expression>
    WHEN { <expression> | <search_condition> } THEN { <expression> |
NULL } [ ... ]
  [ ELSE { <expression> | NULL } ]
  END
  | <literal>
  | <sql_function>

```



```

<search_condition> ::=
{
    <expression> { = | > | < | >= | <= | <> | != | LIKE | IN | NOT IN
| AND | OR | IS NULL | IS NOT NULL } [ <expression> ]
} [ { AND | OR } ... ]

```

Examples

1. Return all columns:

```
SELECT * FROM Customers
```

2. Rename a column:

```
SELECT "CompanyName" AS MY_CompanyName FROM Customers
```

3. Cast a column's data as a different data type:

```
SELECT CAST(Balance AS VARCHAR) AS Str_Balance FROM Customers
```

4. Search data:

```
SELECT * FROM Customers WHERE Country = 'US'
```

5. The SparkSQL APIs support the following operators in the WHERE clause: =, >, <, >=, <=, <>, !=, LIKE, IN, NOT IN, AND, OR, IS NULL, IS NOT NULL.

```
SELECT * FROM Customers WHERE Country = 'US';
```

6. Return the number of items matching the query criteria:

```
SELECT COUNT(*) AS MyCount FROM Customers
```

7. Return the number of unique items matching the query criteria:

```
SELECT COUNT(DISTINCT CompanyName) FROM Customers
```

8. Return the unique items matching the query criteria:

```
SELECT DISTINCT CompanyName FROM Customers
```

9. Summarize data:

```
SELECT CompanyName, MAX(Balance) FROM Customers GROUP BY  
CompanyName
```

See [Aggregate Functions](#) for details.

10. Retrieve data from multiple tables.

```
SELECT "restaurants"."restaurant_id", "restaurants".name,  
"restaurants.grades".* FROM "restaurants.grades" JOIN "restaurants"  
WHERE "restaurants".name = 'Morris Park Bake Shop'
```

See [JOIN Queries](#) for details.

11. Sort a result set in ascending order:

```
SELECT City, CompanyName FROM Customers ORDER BY CompanyName ASC
```

Aggregate Functions

Examples of Aggregate Functions

Below are several examples of SQL aggregate functions. You can use these with a GROUP BY clause to aggregate rows based on the specified GROUP BY criterion. This can be a reporting tool.

COUNT

Returns the number of rows matching the query criteria.

```
SELECT COUNT(*) FROM Customers WHERE Country = 'US'
```

COUNT(DISTINCT)

Returns the number of distinct, non-null field values matching the query criteria.

```
SELECT COUNT(DISTINCT City) AS DistinctValues FROM Customers WHERE  
Country = 'US'
```

AVG

Returns the average of the column values.

```
SELECT CompanyName, AVG(Balance) FROM Customers WHERE Country = 'US'  
GROUP BY CompanyName
```

MIN

Returns the minimum column value.

```
SELECT MIN(Balance), CompanyName FROM Customers WHERE Country = 'US'  
GROUP BY CompanyName
```

MAX

Returns the maximum column value.

```
SELECT CompanyName, MAX(Balance) FROM Customers WHERE Country = 'US'  
GROUP BY CompanyName
```

SUM

Returns the total sum of the column values.

```
SELECT SUM(Balance) FROM Customers WHERE Country = 'US'
```

JOIN Queries

The SparkSQL Adapter supports joins of a nested array with its parent document .

Projection Functions

ROUND(expr [, d])

Returns expr rounded to d decimal places using HALF_UP rounding mode.

- **expr**: Any numeric expression.
- **d**: The number of decimal places.

BROUND(expr [, d])

Returns expr rounded to d decimal places using HALF_EVEN rounding mode.

- **expr**: Any numeric expression.
- **d**: The number of decimal places.

FLOOR(expr)

Returns the largest integer not greater than expr.

- **expr**: Any numeric expression.

CEIL(expr)

Returns the smallest integer not smaller than expr.

- **expr**: Any numeric expression.

RAND([seed])

Returns a random value with independent and identically distributed (i.i.d.) uniformly distributed values in [0, 1).

- **seed**: The seed to use to generate the random value.

EXP(expr)

Returns e to the power of expr.

- **expr**: Any numeric expression.

LN(expr)

Returns the natural logarithm (base e) of expr.

- **expr**: Any numeric expression.

LOG10(expr)

Returns the logarithm of expr with base 10.

- **expr**: Any numeric expression.

LOG2(expr)

Returns the logarithm of expr with base 2.

- **expr**: Any numeric expression.

LOG(base, expr)

Returns the logarithm of expr with base.

- **base**: A numeric expression to use as the base.
- **expr**: Any numeric expression.

POW(expr1, expr2)

Raises expr1 to the power of expr2.

- **expr1**: Any numeric expression.
- **expr2**: Any numeric expression.

SQRT(expr)

Returns the square root of expr.

- **expr**: Any numeric expression.

BIN(expr)

Returns the string representation of the long value `expr` represented in binary.

- **expr**: A long expression.

HEX(expr)

Converts `expr` to hexadecimal.

- **expr**: The expression to convert to hex.

UNHEX(expr)

Converts hexadecimal `expr` to binary.

- **expr**: The hexadecimal value to convert to binary.

CONV(num, from_base, to_base)

Convert `num` from `from_base` to `to_base`.

- **num**: The number to convert.
- **from_base**: The original base of `num`.
- **to_base**: The base to convert `num` to.

ABS(expr)

Returns the absolute value of the numeric value.

- **expr**: Any valid numeric expression.

PMOD(expr1, expr2)

Returns the positive value of `expr1 mod expr2`.

- **expr1**: Any valid numeric expression.
- **expr2**: Any valid numeric expression.

SIN(expr)

Returns the sine of expr, as if computed by `java.lang.Math.sin`.

- **expr**: Any valid numeric expression.

ASIN(expr)

Returns the inverse sine (a.k.a. arc sine) the arc sin of expr, as if computed by `java.lang.Math.asin`.

- **expr**: Any valid numeric expression.

COS(expr)

Returns the cosine of expr, as if computed by `java.lang.Math.cos`.

- **expr**: Any valid numeric expression.

ACOS(expr)

Returns the inverse cosine (a.k.a. arc cosine) of expr, as if computed by `java.lang.Math.acos`.

- **expr**: Any valid numeric expression.

TAN(expr)

Returns the tangent of expr, as if computed by `java.lang.Math.tan`.

- **expr**: Any valid numeric expression.

ATAN(expr)

Returns the inverse tangent (a.k.a. arc tangent) of expr, as if computed by `java.lang.Math.atan`

- **expr**: Any valid numeric expression.

DEGREES(expr)

Converts radians to degrees.

- **expr**: Any valid numeric expression.

RADIANS(expr)

Converts degrees to radians.

- **expr**: Any valid numeric expression.

POSITIVE(expr)

Returns the positive value of expr.

- **expr**: Any valid numeric expression.

NEGATIVE(expr)

Returns the negated value of expr.

- **expr**: Any valid numeric expression.

SIGN(expr)

Returns -1.0, 0.0 or 1.0 as expr is negative, 0 or positive.

- **expr**: Any valid numeric expression.

E()

Returns Euler's number, e.

PI()

Returns pi.

FACTORIAL(expr)

Returns the factorial of expr. expr is [0..20]. Otherwise, null.

- **expr**: A numeric expression.

CBRT(expr)

Returns the cube root of expr.

- **expr**: Any valid numeric expression.

SHIFTELEFT(base, shift)

Bitwise left shift.

- **base**: The base number to shift.
- **shift**: The number of bits to shift.

SHIFTRIGHT(base, shift)

Bitwise right shift.

- **base**: The base number to shift.
- **shift**: The number of bits to shift.

SHIFTRIGHTUNSIGNED(base, shift)

Bitwise unsigned right shift.

- **base**: The base number to shift.
- **shift**: The number of bits to shift.

GREATEST(expr1, expr2 [, expr3] [, ...])

Returns the greatest value of all parameters, skipping null values.

- **expr1**: Any valid expression.
- **expr2**: Any valid expression.
- **expr3**: Any valid expression.

LEAST(expr1, expr2 [, expr3] [, ...])

Returns the least value of all parameters, skipping null values.

- **expr1**: Any valid expression.
- **expr2**: Any valid expression.
- **expr3**: Any valid expression.

WIDTH_BUCKET(expr, min_value, max_value, num_buckets)

Returns an integer between 0 and num_buckets+1 by mapping expr into the ith equally sized bucket. Buckets are made by dividing [min_value, max_value] into equally sized regions. If expr < min_value, return 1, if expr > max_value return num_buckets+1.

- **expr**: A valid numeric expression.
- **min_value**: The minimum value.
- **max_value**: The maximum value.
- **num_buckets**: The number of buckets.

SIZE(expr)

Returns the size of an array or a map. Returns -1 if null.

- **expr**: Any valid expression.

MAP_KEYS(map)

Returns an unordered array containing the keys of the map.

- **map**: A valid map expression.

MAP_VALUES(map)

Returns an unordered array containing the values of the map.

- **map**: A valid map expression.

ARRAY_CONTAINS(array, expr)

Returns true if the array contains the value.

- **array**: The array to search.
- **expr**: The expression to search for.

SORT_ARRAY(array [, ascendingOrder])

Sorts the input array in ascending or descending order according to the natural ordering of the array elements.

- **array**: The array to sort.
- **order**: Identifies whether to sort in ascending order.

BINARY(expr)

Casts the value expr to the target data type binary.

- **expr**: The expression to cast.

CAST(expr AS type)

Casts the value expr to the target data type type.

- **expr:** Any valid expression.
- **type:** The type to cast expr to.

FROM_UNIXTIME(unixtime [, format])

Converts the number of seconds from unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the format of "1970-01-01 00:00:00".

- **unixtime:** Unix time.
- **format:** The format to convert unixtime to.

UNIX_TIMESTAMP([expr [, pattern]])

Returns the UNIX timestamp of the given time.

- **expr:** The time string to convert.
- **format:** The format of expr.

TO_DATE(date_str [, fmt])

Parses the date_str expression with the fmt expression to a date. Returns null with invalid input. By default, it follows casting rules to a date if the fmt is omitted.

- **date_str:** The date string expression.
- **fmt:** The format of date_str.

YEAR(date)

Returns the year component of the date/timestamp.

- **date:** The date to extract the year from.

QUARTER(date)

Returns the quarter of the year for date, in the range 1 to 4.

- **date:** The date to extract the quarter from.

MONTH(date)

Returns the month component of the date/timestamp.

- **date:** The date to extract the month from.

DAY(date)

Returns the day of month of the date/timestamp.

- **date:** The date to extract the day from.

HOUR(timestamp)

Returns the hour component of the string/timestamp.

- **timestamp:** The timestamp to extract the hours from.

MINUTE(timestamp)

Returns the minute component of the string/timestamp.

- **timestamp:** The timestamp to extract the minutes from.

SECOND(timestamp)

Returns the second component of the string/timestamp.

- **timestamp:** The timestamp to extract the seconds from.

WEEKOFYEAR(date)

Returns the week of the year of the given date. A week is considered to start on a Monday and week 1 is the first week with >3 days.

- **date:** The date to extract the week of the year from.

DATEDIFF(endDate, startDate)

Returns the number of days from startDate to endDate.

- **endDate:** The end date.
- **startDate:** The start date.

DATE_ADD(start_date, num_days)

Returns the date that is num_days after start_date.

- **start_date:** The start date.
- **num_days:** The number of days to add to start_date.

DATE_SUB(start_date, num_days)

Returns the date that is num_days before start_date.

- **start_date:** The start date.
- **num_days:** The number of days to subtract from start_date.

FROM_UTC_TIMESTAMP(timestamp, timezone)

Given a timestamp like '2017-07-14 02:40:00.0', interprets it as a time in UTC, and renders that time as a timestamp in the given time zone. For example, 'GMT+1' would yield '2017-07-14 03:40:00.0'.

- **timestamp:** The UTC timestamp.
- **timezone:** The timezone to convert to.

TO_UTC_TIMESTAMP(timestamp, timezone)

Given a timestamp like '2017-07-14 02:40:00.0', interprets it as a time in the given time zone, and renders that time as a timestamp in UTC. For example, 'GMT+1' would yield '2017-07-14 01:40:00.0'.

- **timestamp:** The timestamp to convert to UTC.

- **timezone:** The timezone of timestamp.

CURRENT_DATE()

Returns the current date at the start of query evaluation.

CURRENT_TIMESTAMP()

Returns the current timestamp at the start of query evaluation.

ADD_MONTHS(start_date, num_months [, fmt])

Returns the date that is num_months after start_date.

- **start_date:** The starting date.
- **num_months:** The number of months to add.
- **fmt:** The output format.

LAST_DAY(date)

Returns the last day of the month which the date belongs to.

- **date:** A valid date expression.

NEXT_DAY(start_date, day_of_week)

Returns the first date which is later than start_date and named as indicated.

- **start_date:** The start date.
- **day_of_week:** The day of week.

TRUNC(date, time_unit)

Returns date with the time portion of the day truncated to the unit specified by the format model fmt. fmt should be one of ["year", "yyyy", "yy", "mon", "month", "mm"]

- **date**: A valid date expression.
- **time_unit**: The time unit.

MONTHS_BETWEEN(timestamp1, timestamp2)

Returns number of months between timestamp1 and timestamp2.

- **timestamp1**: A valid timestamp expression.
- **timestamp2**: A valid timestamp expression.

DATE_FORMAT(timestamp, fmt)

Converts timestamp to a value of string in the format specified by the date format fmt.

- **timestamp**: A valid timestamp expression.
- **fmt**: A valid date format.

IF(expr1, expr2, expr3)

If expr1 evaluates to true, then returns expr2; otherwise returns expr3.

- **expr1**: An expression that should evaluate to a boolean value.
- **expr2**: A valid expression.
- **expr3**: A valid expression.

ISNULL(expr)

Returns true if expr is null, or false otherwise.

- **expr**: A valid expression.

ISNOTNULL(expr)

Returns true if expr is not null, or false otherwise.

- **expr**: A valid expression.

NVL(expr1, expr2)

Returns expr1 if it's not NaN, or expr2 otherwise.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

COALESCE(expr1, expr2 [, expr3] [, ...])

Returns the first non-null argument if exists. Otherwise, null.

- **expr1**: A valid expression.
- **expr2**: A valid expression.
- **expr3**: A valid expression.

NULLIF(expr1, expr2)

Returns null if expr1 equals to expr2, or expr1 otherwise.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

ASSERT_TRUE(expr)

Throws an exception if expr is not true.

- **expr**: A valid expression that evaluates to a boolean.

ASCII(str)

Returns the numeric value of the first character of str.

- **str**: A string expression.

BASE64(bin)

Converts the argument from a binary bin to a base 64 string.

- **bin**: A binary expression.

CHAR_LENGTH(str)

Returns the character length of string data or number of bytes of binary data. The length of string data includes the trailing spaces. The length of binary data includes binary zeros.

- **str**: A string expression.

CHR(expr)

Returns the ASCII character having the binary equivalent to expr. If n is larger than 256 the result is equivalent to chr(n % 256)

- **expr**: A integer expression.

CONCAT(str1, str2 [, str3] [, ...])

Returns the string or bytes resulting from concatenating the strings or bytes passed in as parameters in order. For example, concat('foo', 'bar') results in 'foobar'. Note that this function can take any number of input strings.

- **str1**: A valid string expression.
- **str2**: A valid string expression.
- **str3**: A valid string expression.

CONCAT_WS(sep [, exp1] [, ...])

Returns the concatenation of the strings separated by sep.

- **set**: A string separator.
- **exp1**: A valid expression.

DECODE(bin, charset)

Decodes the first argument using the second argument character set.

- **bin**: The binary expression to decode.
- **charset**: The charset to use to decode bin.

ELT(n, input1 [, input2] [, ...])

Returns the n-th input, e.g., returns input2 when n is 2.

- **n**: A valid integer index.
- **input1**: A valid string expression.
- **input3**: A valid string expression.

ENCODE(str, charset)

Encodes the first argument using the second argument character set.

- **str**: A string expression to encode.
- **charset**: The charset to use to encode str.

FIELD(val1, val2 [, val3] [, ...])

Returns the index of val in the val1,val2,val3,... list or 0 if not found. For example field ('world','say','hello','world') returns 3. All primitive types are supported, arguments are compared using str.equals(x). If val is NULL, the return value is 0.

- **val1**: A valid expression.
- **val2**: A valid expression.
- **val3**: A valid expression.

FIND_IN_SET(str, str_array)

Returns the index (1-based) of the given string (str) in the comma-delimited list (str_array). Returns 0, if the string was not found or if the given string (str) contains a comma.

- **str**: The string expression to search for.
- **str_array**: A comma-delimited list of values.

FORMAT_NUMBER(expr1, expr2)

Formats the number expr1 like '#,###,###.##', rounded to expr2 decimal places. If expr2 is 0, the result has no decimal point or fractional part. This is supposed to function like MySQL's FORMAT.

- **expr1**: A numeric expression to format.
- **expr2**: The number of decimal places.

GET_JSON_OBJECT(json_txt, path)

Extracts a json object from path.

- **json_txt**: JSON data.
- **path**: The path to extract.

IN_FILE(str, filename)

Returns true if the string str appears as an entire line in filename.

- **str**: The string to search for.
- **filename**: The name of the file to search.

INSTR(str, substr)

Returns the (1-based) index of the first occurrence of substr in str.

- **str**: A string expression.
- **substr**: The string expression to search for.

LENGTH(expr)

Returns the character length of string data or number of bytes of binary data. The length of string data includes the trailing spaces. The length of binary data includes binary zeros.

- **expr**: A string expression.

LOCATE(substr, str [, pos])

Returns the position of the first occurrence of substr in str after position pos. The given pos and return value are 1-based.

- **substr**: The string expression to search for.
- **str**: The string expression to search in.
- **pos**: The starting index.

LOWER(expr)

Returns str with all characters changed to lowercase.

- **expr**: A string expression.

LPAD(str, len, pad_str)

Returns str, left-padded with pad to a length of len. If str is longer than len, the return value is shortened to len characters.

- **str**: A string expression.
- **len**: The length to pad.
- **pad_str**: The pad string.

LTRIM(str)

Removes the leading space characters from str.

- **str**: A string expression.

OCTET_LENGTH(expr)

Returns the byte length of expr or number of bytes in binary data.

- **expr**: Any string expression.

PARSE_URL(url, partToExtract [, key])

Returns the specified part from the URL. For example, `parse_url('http://facebook.com/path1/p.php?k1=v1#Ref1', 'HOST')` returns 'facebook.com'. Also a value of a particular key in QUERY can be extracted by providing the key as the third argument, for example, `parse_url('http://facebook.com/path1/p.php?k1=v1#Ref1', 'QUERY', 'k1')` returns 'v1'.

- **url:** A valid URL expression.
- **partToExtract:** The URL part to extract. Valid values for `partToExtract` include HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, and USERINFO.
- **key:** The key.

PRINTF(strfmt [, obj1] [, ...])

Returns a formatted string from printf-style format strings.

- **strfmt:** The string format.
- **obj1:** The object to include in the formatted string.

REGEXP_EXTRACT(str, regexp [, idx])

Extracts a group that matches regexp.

- **str:** A string expression.
- **regexp:** A regular expression to search for.
- **idx:** The starting index.

REGEXP_REPLACE(str, regexp, rep)

Replaces all substring of str that match regexp with rep.

- **str:** A string expression.
- **regexp:** A regular expression to search for.
- **rep:** The string to replace.

REPEAT(str, n)

Returns the string which repeats the given string value n times.

- **str**: The string expression to repeat.
- **n**: The number of times to repeat str.

REPLACE(str, search [, replace])

Replaces all occurrences of search with replace. If search is not found in str, str is returned unchanged. If replace is not specified or is an empty string, nothing replaces the string that is removed from str.

- **str**: A string expression.
- **search**: The search string.
- **replace**: A string expression to replace search values.

REVERSE(str)

Returns the reversed given string.

- **str**: A string expression.

RPAD(str, len, pad_str)

Returns str, right-padded with pad to a length of len. If str is longer than len, the return value is shortened to len characters.

- **str**: A string expression.
- **len**: The length to pad.
- **pad_str**: The pad string.

RTRIM(str)

Removes the trailing space characters from str.

- **str**: A string expression.

SENTENCES(str [, lang, country])

Splits str into an array of array of words.

- **str**: A string expression.
- **lang**: The language of str.
- **country**: The country of the specified language.

SPACE(n)

Returns a string consisting of n spaces.

- **n**: The number of spaces.

SPLIT(str, regex)

Splits str around occurrences that match regex.

- **str**: A string expression.
- **regex**: The regular expression to match.

STR_TO_MAP(text [, pairDelim [, keyValueDelim]])

Creates a map after splitting the text into key/value pairs using delimiters. Default delimiters are ',' for pairDelim and ':' for keyValueDelim.

- **text**: A string expression.
- **pairDelim**: The pair delimiter.
- **keyValueDelim**: The value delimiter.

SUBSTR(str, pos [, len])

Returns the substring of str that starts at pos and is of length len, or the slice of byte array that starts at pos and is of length len.

- **str**: A string expression.
- **pos**: The starting position.
- **len**: The length of the string.

SUBSTRING_INDEX(str, delim, count)

Returns the substring from str before count occurrences of the delimiter delim. If count is positive, everything to the left of the final delimiter (counting from the left) is returned. If count is negative, everything to the right of the final delimiter (counting from the right) is returned. The function substring_index performs a case-sensitive match when searching for delim.

- **str**: A string expression.
- **delim**: The delimiter.
- **count**: Total number of occurrences.

TRANSLATE(input, from, to)

Translates the input string by replacing the characters present in the from string with the corresponding characters in the to string.

- **input**: A string expression.
- **from**: A string expression.
- **to**: A string expression.

TRIM(str)

Removes the leading and trailing space characters from str.

- **str**: A string expression.

UNBASE64(str)

Converts the argument from a base 64 string str to a binary.

- **str**: A string expression.

UPPER(str)

Returns str with all characters changed to uppercase.

- **str**: A string expression.

INITCAP(str)

Returns str with the first letter of each word in uppercase. All other letters are in lowercase. Words are delimited by white space.

- **str**: A string expression.

LEVENSHTEIN(str1, str2)

Returns the Levenshtein distance between the two given strings.

- **str1**: A string expression.
- **str2**: A string expression.

SOUNDEX(str)

Returns Soundex code of the string.

- **str**: A string expression.

MASK(str [, upper [, lower [, number]]])

Returns a masked version of str. By default, upper case letters are converted to "X", lower case letters are converted to "x" and numbers are converted to "n". For example mask("abcd-EFGH-8765-4321") results in xxxx-XXXX-nnnn-nnnn. You can override the characters used in the mask by supplying additional arguments: the second argument controls the mask character for upper case letters, the third argument for lower case letters and the fourth argument for numbers. For example, mask("abcd-EFGH-8765-4321", "U", "l", "#") results in llll-UUUU-####-####.

- **str**: The string to mask.
- **upper**: The character to mask for uppercase letters.

- **lower:** The character to mask for lowercase letters.
- **number:** The character to mask for numbers.

MASK_FIRST_N(str [, n])

Returns a masked version of str with the first n values masked. Upper case letters are converted to "X", lower case letters are converted to "x" and numbers are converted to "n". For example, mask_first_n("1234-5678-8765-4321", 4) results in nnnn-5678-8765-4321.

- **str:** The string to mask.
- **n:** The number of values to mask.

MASK_LAST_N(str [, n])

Returns a masked version of str with the last n values masked. Upper case letters are converted to "X", lower case letters are converted to "x" and numbers are converted to "n". For example, mask_last_n("1234-5678-8765-4321", 4) results in 1234-5678-8765-nnnn.

- **str:** The string to mask.
- **n:** The number of values to mask.

MASK_SHOW_FIRST_N(str [, n])

Returns a masked version of str, showing the first n characters unmasked. Upper case letters are converted to "X", lower case letters are converted to "x" and numbers are converted to "n". For example, mask_show_first_n("1234-5678-8765-4321", 4) results in 1234-nnnn-nnnn-nnnn.

- **str:** The string to mask.
- **n:** The number of values to mask.

MASK_SHOW_LAST_N(str [, n])

Returns a masked version of str, showing the last n characters unmasked. Upper case letters are converted to "X", lower case letters are converted to "x" and numbers are converted to "n". For example, mask_show_last_n("1234-5678-8765-4321", 4) results in nnnn-nnnn-nnnn-4321.

- **str**: The string to mask.
- **n**: The number of values to mask.

MASK_HASH(str)

Returns a hashed value based on str. The hash is consistent and can be used to join masked values together across tables. This function returns null for non-string types.

- **str**: The string to mask.

JAVA_METHOD(class, method [, arg1] [, ...])

Calls a method with reflection.

- **class**: The class to call.
- **method**: The method to call.
- **arg1**: The argument to pass in.

REFLECT(class, method [, arg1] [, ...])

Calls a method with reflection.

- **class**: The class to call.
- **method**: The method to call.
- **arg1**: The argument to pass in.

HASH(expr1 [, expr2] [, ...])

Returns a hash value of the arguments.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

CURRENT_USER()

Returns current user name from the configured authenticator manager. Could be the same as the user provided when connecting, but with some authentication managers (for example HadoopDefaultAuthenticator) it could be different.

LOGGED_IN_USER()

Returns current user name from the session state. This is the username provided when connecting to Hive.

CURRENT_DATABASE()

Returns current database name.

SHA1(expr)

Returns a sha1 hash value as a hex string of the expr.

- **expr**: A valid expression.

CRC32(expr)

Returns a cyclic redundancy check value of the expr as a bigint.

- **expr**: A valid expression.

SHA2(expr, bitlength)

Returns a checksum of SHA-2 family as a hex string of expr. SHA-224, SHA-256, SHA-384, and SHA-512 are supported. Bit length of 0 is equivalent to 256.

- **expr**: A valid expression.
- **bitlength**: The bit length.

AES_ENCRYPT(input, key)

Encrypt input using AES. Key lengths of 128, 192 or 256 bits can be used. 192 and 256 bits keys can be used if Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files are installed. If either argument is NULL or the key length is not one of the permitted values, the return value is NULL. Example: `base64(aes_encrypt('ABC', '1234567890123456')) = 'y6Ss+zCYObpCbgfWfyNWTw=='`.

- **input:** The input value to encrypt.
- **key:** The key to use when encrypting.

VERSION()

Returns the Hive version. The string contains 2 fields, the first being a build number and the second being a build hash. Example: `"select version();"` might return `"2.1.0.2.5.0.0-1245r027527b9c5ce1a3d7d0b6d2e6de2378fb0c39232"`. Actual results will depend on your build.

COUNT(DISTINCT expr1 [, expr2] [, ...])

Returns the number of rows for which the supplied expression(s) are unique and non-null.

- **expr1:** A valid expression.
- **expr2:** A valid expression.

SUM(expr)

Returns the sum calculated from values of a group.

- **expr:** A valid expression.

SUM(DISTINCT expr)

Returns the sum calculated from distinct values of a group.

- **expr:** A valid expression.

AVG(expr)

Returns the mean calculated from values of a group.

- **expr**: A valid expression.

AVG(DISTINCT expr)

Returns the mean calculated from distinct values of a group.

- **expr**: A valid expression.

MIN(expr)

Returns the minimum value of expr.

- **expr**: A valid expression.

MAX(expr)

Returns the maximum value of expr.

- **expr**: A valid expression.

VARIANCE(expr)

Returns the sample variance calculated from values of a group.

- **expr**: A valid expression.

STDDEV_POP(expr)

Returns the population standard deviation calculated from values of a group.

- **expr**: A valid expression.

STDDEV_SAMP(expr)

Returns the sample standard deviation calculated from values of a group.

- **expr**: A valid expression.

COVAR_POP(expr1, expr2)

Returns the population covariance of a set of number pairs.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

COVAR_SAMP(expr1, expr2)

Returns the sample covariance of a set of number pairs.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

CORR(expr1, expr2)

Returns Pearson coefficient of correlation between a set of number pairs.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

PERCENTILE(col, percentage [, accuracy])

Returns the exact percentile value of numeric column col at the given percentage. The value of percentage must be between 0.0 and 1.0. The value of frequency should be positive integral

- **col**: A numeric expression.
- **percentage**: The percentage.
- **accuracy**: The accuracy to control approximation.

PERCENTILE_APPROX(col, percentage [, accuracy])

Returns the approximate percentile value of numeric column `col` at the given percentage. The value of percentage must be between 0.0 and 1.0. The accuracy parameter (default: 10000) is a positive numeric literal which controls approximation accuracy at the cost of memory. Higher value of accuracy yields better accuracy, $1.0/\text{accuracy}$ is the relative error of the approximation. When percentage is an array, each value of the percentage array must be between 0.0 and 1.0. In this case, returns the approximate percentile array of column `col` at the given percentage array.

- **col**: A numeric expression.
- **percentage**: The percentage.
- **accuracy**: The accuracy to control approximation.

COLLECT_SET(expr)

Collects and returns a set of unique elements.

- **expr**: A valid expression.

COLLECT_LIST(expr)

Collects and returns a set of unique elements.

- **expr**: A valid expression.

NTILE(n)

Divides the rows for each window partition into `n` buckets ranging from 1 to at most `n`.

- **n**: The number of buckets.

EXPLODE(expr)

Separates the elements of array `expr` into multiple rows, or the elements of map `expr` into multiple rows and columns.

- **expr**: A valid expression.

POSEXPLODE(expr)

Separates the elements of array `expr` into multiple rows with positions, or the elements of map `expr` into multiple rows and columns with positions.

- **expr**: A valid expression.

INLINE(expr)

Explodes an array of structs into a table.

- **expr**: A valid expression.

STACK(n, expr1 [, expr2] [, ...])

Separates `expr1`, ..., `exprk` into `n` rows.

- **n**: The number of rows.
- **expr1**: A valid expression.
- **expr2**: A valid expression.

PARSE_URL_TUPLE(urlStr, p1 [, p2] [, ...])

Takes URL string and a set of `n` URL parts, and returns a tuple of `n` values. This is similar to the `parse_url()` UDF but can extract multiple parts at once out of a URL. Valid part names are: HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, USERINFO, QUERY:[KEY].

- **urlStr**: A valid URL string.
- **p1**: A valid part name.
- **p2**: A valid part name.

JSON_EXTRACT(json, jsonpath)

Selects any value in a JSON array or object. The path to the array is specified in the `jsonpath` argument. Return value is numeric or null.

- **json**: The JSON document to extract.

- **jsonpath:** The XPath used to select the nodes. The JSONPath must be a string constant. The values of the nodes selected will be returned in a token-separated list.

XML_EXTRACT(xml, xpath [, separator])

Extracts an XML document using the specified XPath to flatten the XML. A comma is used to separate the outputs by default, but this can be changed by specifying the third parameter.

- **xml:** The XML document to extract.
- **xpath:** The XPath used to select the nodes. The nodes selected will be returned in a token-separated list.
- **separator:** The optional token used to separate the items in the flattened response. If this is not specified, the separator will be a comma.

Predicate Functions

ROUND(expr [, d])

Returns expr rounded to d decimal places using HALF_UP rounding mode.

- **expr:** Any numeric expression.
- **d:** The number of decimal places.

BROUND(expr [, d])

Returns expr rounded to d decimal places using HALF_EVEN rounding mode.

- **expr:** Any numeric expression.
- **d:** The number of decimal places.

FLOOR(expr)

Returns the largest integer not greater than expr.

- **expr:** Any numeric expression.

CEIL(expr)

Returns the smallest integer not smaller than expr.

- **expr**: Any numeric expression.

RAND([seed])

Returns a random value with independent and identically distributed (i.i.d.) uniformly distributed values in [0, 1).

- **seed**: The seed to use to generate the random value.

EXP(expr)

Returns e to the power of expr.

- **expr**: Any numeric expression.

LN(expr)

Returns the natural logarithm (base e) of expr.

- **expr**: Any numeric expression.

LOG10(expr)

Returns the logarithm of expr with base 10.

- **expr**: Any numeric expression.

LOG2(expr)

Returns the logarithm of expr with base 2.

- **expr**: Any numeric expression.

LOG(base, expr)

Returns the logarithm of expr with base.

- **base**: A numeric expression to use as the base.
- **expr**: Any numeric expression.

POW(expr1, expr2)

Raises expr1 to the power of expr2.

- **expr1**: Any numeric expression.
- **expr2**: Any numeric expression.

SQRT(expr)

Returns the square root of expr.

- **expr**: Any numeric expression.

BIN(expr)

Returns the string representation of the long value expr represented in binary.

- **expr**: A long expression.

HEX(expr)

Converts expr to hexadecimal.

- **expr**: The expression to convert to hex.

UNHEX(expr)

Converts hexadecimal expr to binary.

- **expr**: The hexadecimal value to convert to binary.

CONV(num, from_base, to_base)

Convert num from from_base to to_base.

- **num**: The number to convert.
- **from_base**: The original base of num.
- **to_base**: The base to convert num to.

ABS(expr)

Returns the absolute value of the numeric value.

- **expr**: Any valid numeric expression.

PMOD(expr1, expr2)

Returns the positive value of expr1 mod expr2.

- **expr1**: Any valid numeric expression.
- **expr2**: Any valid numeric expression.

SIN(expr)

Returns the sine of expr, as if computed by java.lang.Math.sin.

- **expr**: Any valid numeric expression.

ASIN(expr)

Returns the inverse sine (a.k.a. arc sine) the arc sin of expr, as if computed by java.lang.Math.asin.

- **expr**: Any valid numeric expression.

COS(expr)

Returns the cosine of expr, as if computed by java.lang.Math.cos.

- **expr**: Any valid numeric expression.

ACOS(expr)

Returns the inverse cosine (a.k.a. arc cosine) of expr, as if computed by `java.lang.Math.acos`.

- **expr**: Any valid numeric expression.

TAN(expr)

Returns the tangent of expr, as if computed by `java.lang.Math.tan`.

- **expr**: Any valid numeric expression.

ATAN(expr)

Returns the inverse tangent (a.k.a. arc tangent) of expr, as if computed by `java.lang.Math.atan`.

- **expr**: Any valid numeric expression.

DEGREES(expr)

Converts radians to degrees.

- **expr**: Any valid numeric expression.

RADIANS(expr)

Converts degrees to radians.

- **expr**: Any valid numeric expression.

POSITIVE(expr)

Returns the positive value of expr.

- **expr**: Any valid numeric expression.

NEGATIVE(expr)

Returns the negated value of expr.

- **expr**: Any valid numeric expression.

SIGN(expr)

Returns -1.0, 0.0 or 1.0 as expr is negative, 0 or positive.

- **expr**: Any valid numeric expression.

E()

Returns Euler's number, e.

PI()

Returns pi.

FACTORIAL(expr)

Returns the factorial of expr. expr is [0..20]. Otherwise, null.

- **expr**: A numeric expression.

CBRT(expr)

Returns the cube root of expr.

- **expr**: Any valid numeric expression.

SHIFTELFT(base, shift)

Bitwise left shift.

- **base**: The base number to shift.
- **shift**: The number of bits to shift.

SHIFTRIGHT(base, shift)

Bitwise right shift.

- **base**: The base number to shift.
- **shift**: The number of bits to shift.

SHIFTRIGHTUNSIGNED(base, shift)

Bitwise unsigned right shift.

- **base**: The base number to shift.
- **shift**: The number of bits to shift.

GREATEST(expr1, expr2 [, expr3] [, ...])

Returns the greatest value of all parameters, skipping null values.

- **expr1**: Any valid expression.
- **expr2**: Any valid expression.
- **expr3**: Any valid expression.

LEAST(expr1, expr2 [, expr3] [, ...])

Returns the least value of all parameters, skipping null values.

- **expr1**: Any valid expression.
- **expr2**: Any valid expression.
- **expr3**: Any valid expression.

WIDTH_BUCKET(expr, min_value, max_value, num_buckets)

Returns an integer between 0 and num_buckets+1 by mapping expr into the ith equally sized bucket. Buckets are made by dividing [min_value, max_value] into equally sized regions. If expr < min_value, return 1, if expr > max_value return num_buckets+1.

- **expr**: A valid numeric expression.
- **min_value**: The minimum value.
- **max_value**: The maximum value.
- **num_buckets**: The number of buckets.

SIZE(expr)

Returns the size of an array or a map. Returns -1 if null.

- **expr**: Any valid expression.

MAP_KEYS(map)

Returns an unordered array containing the keys of the map.

- **map**: A valid map expression.

MAP_VALUES(map)

Returns an unordered array containing the values of the map.

- **map**: A valid map expression.

ARRAY_CONTAINS(array, expr)

Returns true if the array contains the value.

- **array**: The array to search.
- **expr**: The expression to search for.

SORT_ARRAY(array [, ascendingOrder])

Sorts the input array in ascending or descending order according to the natural ordering of the array elements.

- **array**: The array to sort.
- **order**: Identifies whether to sort in ascending order.

BINARY(expr)

Casts the value expr to the target data type binary.

- **expr**: The expression to cast.

CAST(expr AS type)

Casts the value expr to the target data type type.

- **expr**: Any valid expression.
- **type**: The type to cast expr to.

FROM_UNIXTIME(unixtime [, format])

Converts the number of seconds from unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the format of "1970-01-01 00:00:00".

- **unixtime**: Unix time.
- **format**: The format to convert unixtime to.

UNIX_TIMESTAMP([expr [, pattern]])

Returns the UNIX timestamp of the given time.

- **expr**: The time string to convert.
- **format**: The format of expr.

TO_DATE(date_str [, fmt])

Parses the date_str expression with the fmt expression to a date. Returns null with invalid input. By default, it follows casting rules to a date if the fmt is omitted.

- **date_str**: The date string expression.
- **fmt**: The format of date_str.

YEAR(date)

Returns the year component of the date/timestamp.

- **date**: The date to extract the year from.

QUARTER(date)

Returns the quarter of the year for date, in the range 1 to 4.

- **date**: The date to extract the quarter from.

MONTH(date)

Returns the month component of the date/timestamp.

- **date**: The date to extract the month from.

DAY(date)

Returns the day of month of the date/timestamp.

- **date**: The date to extract the day from.

HOUR(timestamp)

Returns the hour component of the string/timestamp.

- **timestamp**: The timestamp to extract the hours from.

MINUTE(timestamp)

Returns the minute component of the string/timestamp.

- **timestamp**: The timestamp to extract the minutes from.

SECOND(timestamp)

Returns the second component of the string/timestamp.

- **timestamp**: The timestamp to extract the seconds from.

WEEKOFYEAR(date)

Returns the week of the year of the given date. A week is considered to start on a Monday and week 1 is the first week with >3 days.

- **date**: The date to extract the week of the year from.

DATEDIFF(endDate, startDate)

Returns the number of days from startDate to endDate.

- **endDate**: The end date.
- **startDate**: The start date.

DATE_ADD(start_date, num_days)

Returns the date that is num_days after start_date.

- **start_date**: The start date.
- **num_days**: The number of days to add to start_date.

DATE_SUB(start_date, num_days)

Returns the date that is num_days before start_date.

- **start_date:** The start date.
- **num_days:** The number of days to subtract from start_date.

FROM_UTC_TIMESTAMP(timestamp, timezone)

Given a timestamp like '2017-07-14 02:40:00.0', interprets it as a time in UTC, and renders that time as a timestamp in the given time zone. For example, 'GMT+1' would yield '2017-07-14 03:40:00.0'.

- **timestamp:** The UTC timestamp.
- **timezone:** The timezone to convert to.

TO_UTC_TIMESTAMP(timestamp, timezone)

Given a timestamp like '2017-07-14 02:40:00.0', interprets it as a time in the given time zone, and renders that time as a timestamp in UTC. For example, 'GMT+1' would yield '2017-07-14 01:40:00.0'.

- **timestamp:** The timestamp to convert to UTC.
- **timezone:** The timezone of timestamp.

CURRENT_DATE()

Returns the current date at the start of query evaluation.

CURRENT_TIMESTAMP()

Returns the current timestamp at the start of query evaluation.

ADD_MONTHS(start_date, num_months [, fmt])

Returns the date that is num_months after start_date.

- **start_date:** The starting date.
- **num_months:** The number of months to add.

- **fmt:** The output format.

LAST_DAY(date)

Returns the last day of the month which the date belongs to.

- **date:** A valid date expression.

NEXT_DAY(start_date, day_of_week)

Returns the first date which is later than start_date and named as indicated.

- **start_date:** The start date.
- **day_of_week:** The day of week.

TRUNC(date, time_unit)

Returns date with the time portion of the day truncated to the unit specified by the format model fmt. fmt should be one of ["year", "yyyy", "yy", "mon", "month", "mm"]

- **date:** A valid date expression.
- **time_unit:** The time unit.

MONTHS_BETWEEN(timestamp1, timestamp2)

Returns number of months between timestamp1 and timestamp2.

- **timestamp1:** A valid timestamp expression.
- **timestamp2:** A valid timestamp expression.

DATE_FORMAT(timestamp, fmt)

Converts timestamp to a value of string in the format specified by the date format fmt.

- **timestamp:** A valid timestamp expression.
- **fmt:** A valid date format.

IF(expr1, expr2, expr3)

If expr1 evaluates to true, then returns expr2; otherwise returns expr3.

- **expr1**: An expression that should evaluate to a boolean value.
- **expr2**: A valid expression.
- **expr3**: A valid expression.

ISNULL(expr)

Returns true if expr is null, or false otherwise.

- **expr**: A valid expression.

ISNOTNULL(expr)

Returns true if expr is not null, or false otherwise.

- **expr**: A valid expression.

NVL(expr1, expr2)

Returns expr1 if it's not NaN, or expr2 otherwise.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

COALESCE(expr1, expr2 [, expr3] [, ...])

Returns the first non-null argument if exists. Otherwise, null.

- **expr1**: A valid expression.
- **expr2**: A valid expression.
- **expr3**: A valid expression.

NULLIF(expr1, expr2)

Returns null if expr1 equals to expr2, or expr1 otherwise.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

ASSERT_TRUE(expr)

Throws an exception if expr is not true.

- **expr**: A valid expression that evaluates to a boolean.

ASCII(str)

Returns the numeric value of the first character of str.

- **str**: A string expression.

BASE64(bin)

Converts the argument from a binary bin to a base 64 string.

- **bin**: A binary expression.

CHAR_LENGTH(str)

Returns the character length of string data or number of bytes of binary data. The length of string data includes the trailing spaces. The length of binary data includes binary zeros.

- **str**: A string expression.

CHR(expr)

Returns the ASCII character having the binary equivalent to expr. If n is larger than 256 the result is equivalent to chr(n % 256)

- **expr**: A integer expression.

CONCAT(str1, str2 [, str3] [, ...])

Returns the string or bytes resulting from concatenating the strings or bytes passed in as parameters in order. For example, `concat('foo', 'bar')` results in 'foobar'. Note that this function can take any number of input strings.

- **str1**: A valid string expression.
- **str2**: A valid string expression.
- **str3**: A valid string expression.

CONCAT_WS(sep [, exp1] [, ...])

Returns the concatenation of the strings separated by sep.

- **sep**: A string separator.
- **exp1**: A valid expression.

DECODE(bin, charset)

Decodes the first argument using the second argument character set.

- **bin**: The binary expression to decode.
- **charset**: The charset to use to decode bin.

ELT(n, input1 [, input2] [, ...])

Returns the n-th input, e.g., returns input2 when n is 2.

- **n**: A valid integer index.
- **input1**: A valid string expression.
- **input3**: A valid string expression.

ENCODE(str, charset)

Encodes the first argument using the second argument character set.

- **str**: A string expression to encode.
- **charset**: The charset to use to encode str.

FIELD(val1, val2 [, val3] [, ...])

Returns the index of val in the val1,val2,val3,... list or 0 if not found. For example field ('world','say','hello','world') returns 3. All primitive types are supported, arguments are compared using str.equals(x). If val is NULL, the return value is 0.

- **val1**: A valid expression.
- **val2**: A valid expression.
- **val3**: A valid expression.

FIND_IN_SET(str, str_array)

Returns the index (1-based) of the given string (str) in the comma-delimited list (str_array). Returns 0, if the string was not found or if the given string (str) contains a comma.

- **str**: The string expression to search for.
- **str_array**: A comma-delimited list of values.

FORMAT_NUMBER(expr1, expr2)

Formats the number expr1 like '#,###,###.##', rounded to expr2 decimal places. If expr2 is 0, the result has no decimal point or fractional part. This is supposed to function like MySQL's FORMAT.

- **expr1**: A numeric expression to format.
- **expr2**: The number of decimal places.

GET_JSON_OBJECT(json_txt, path)

Extracts a json object from path.

- **json_txt**: JSON data.
- **path**: The path to extract.

IN_FILE(str, filename)

Returns true if the string str appears as an entire line in filename.

- **str**: The string to search for.
- **filename**: The name of the file to search.

INSTR(str, substr)

Returns the (1-based) index of the first occurrence of substr in str.

- **str**: A string expression.
- **substr**: The string expression to search for.

LENGTH(expr)

Returns the character length of string data or number of bytes of binary data. The length of string data includes the trailing spaces. The length of binary data includes binary zeros.

- **expr**: A string expression.

LOCATE(substr, str [, pos])

Returns the position of the first occurrence of substr in str after position pos. The given pos and return value are 1-based.

- **substr**: The string expression to search for.
- **str**: The string expression to search in.
- **pos**: The starting index.

LOWER(expr)

Returns str with all characters changed to lowercase.

- **expr**: A string expression.

LPAD(str, len, pad_str)

Returns str, left-padded with pad to a length of len. If str is longer than len, the return value is shortened to len characters.

- **str**: A string expression.
- **len**: The length to pad.
- **pad_str**: The pad string.

LTRIM(str)

Removes the leading space characters from str.

- **str**: A string expression.

OCTET_LENGTH(expr)

Returns the byte length of expr or number of bytes in binary data.

- **expr**: Any string expression.

PARSE_URL(url, partToExtract [, key])

Returns the specified part from the URL. For example, `parse_url('http://facebook.com/path1/p.php?k1=v1#Ref1', 'HOST')` returns 'facebook.com'. Also a value of a particular key in QUERY can be extracted by providing the key as the third argument, for example, `parse_url('http://facebook.com/path1/p.php?k1=v1#Ref1', 'QUERY', 'k1')` returns 'v1'.

- **url**: A valid URL expression.
- **partToExtract**: The URL part to extract. Valid values for partToExtract include HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, and USERINFO.
- **key**: The key.

PRINTF(strfmt [, obj1] [, ...])

Returns a formatted string from printf-style format strings.

- **strfmt**: The string format.
- **obj1**: The object to include in the formatted string.

REGEXP_EXTRACT(str, regexp [, idx])

Extracts a group that matches regexp.

- **str**: A string expression.
- **regexp**: A regular expression to search for.
- **idx**: The starting index.

REGEXP_REPLACE(str, regexp, rep)

Replaces all substring of str that match regexp with rep.

- **str**: A string expression.
- **regexp**: A regular expression to search for.
- **rep**: The string to replace.

REPEAT(str, n)

Returns the string which repeats the given string value n times.

- **str**: The string expression to repeat.
- **n**: The number of times to repeat str.

REPLACE(str, search [, replace])

Replaces all occurrences of search with replace. If search is not found in str, str is returned unchanged. If replace is not specified or is an empty string, nothing replaces the string that is removed from str.

- **str**: A string expression.
- **search**: The search string.
- **replace**: A string expression to replace search values.

REVERSE(str)

Returns the reversed given string.

- **str**: A string expression.

RPAD(str, len, pad_str)

Returns str, right-padded with pad to a length of len. If str is longer than len, the return value is shortened to len characters.

- **str**: A string expression.
- **len**: The length to pad.
- **pad_str**: The pad string.

RTRIM(str)

Removes the trailing space characters from str.

- **str**: A string expression.

SENTENCES(str [, lang, country])

Splits str into an array of array of words.

- **str**: A string expression.
- **lang**: The language of str.
- **country**: The country of the specified language.

SPACE(n)

Returns a string consisting of n spaces.

- **n**: The number of spaces.

SPLIT(str, regex)

Splits str around occurrences that match regex.

- **str**: A string expression.
- **regex**: The regular expression to match.

STR_TO_MAP(text [, pairDelim [, keyValueDelim]])

Creates a map after splitting the text into key/value pairs using delimiters. Default delimiters are ',' for pairDelim and ':' for keyValueDelim.

- **text**: A string expression.
- **pairDelim**: The pair delimiter.
- **keyValueDelim**: The value delimiter.

SUBSTR(str, pos [, len])

Returns the substring of str that starts at pos and is of length len, or the slice of byte array that starts at pos and is of length len.

- **str**: A string expression.
- **pos**: The starting position.
- **len**: The length of the string.

SUBSTRING_INDEX(str, delim, count)

Returns the substring from str before count occurrences of the delimiter delim. If count is positive, everything to the left of the final delimiter (counting from the left) is returned. If count is negative, everything to the right of the final delimiter (counting from the right) is returned. The function substring_index performs a case-sensitive match when searching for delim.

- **str**: A string expression.
- **delim**: The delimiter.

- **count:** Total number of occurrences.

TRANSLATE(input, from, to)

Translates the input string by replacing the characters present in the from string with the corresponding characters in the to string.

- **input:** A string expression.
- **from:** A string expression.
- **to:** A string expression.

TRIM(str)

Removes the leading and trailing space characters from str.

- **str:** A string expression.

UNBASE64(str)

Converts the argument from a base 64 string str to a binary.

- **str:** A string expression.

UPPER(str)

Returns str with all characters changed to uppercase.

- **str:** A string expression.

INITCAP(str)

Returns str with the first letter of each word in uppercase. All other letters are in lowercase. Words are delimited by white space.

- **str:** A string expression.

LEVENSHTEIN(str1, str2)

Returns the Levenshtein distance between the two given strings.

- **str1**: A string expression.
- **str2**: A string expression.

SOUNDEX(str)

Returns Soundex code of the string.

- **str**: A string expression.

MASK(str [, upper [, lower [, number]]])

Returns a masked version of str. By default, upper case letters are converted to "X", lower case letters are converted to "x" and numbers are converted to "n". For example mask ("abcd-EFGH-8765-4321") results in xxxx-XXXX-nnnn-nnnn. You can override the characters used in the mask by supplying additional arguments: the second argument controls the mask character for upper case letters, the third argument for lower case letters and the fourth argument for numbers. For example, mask("abcd-EFGH-8765-4321", "U", "l", "#") results in llll-UUUU-####-####.

- **str**: The string to mask.
- **upper**: The character to mask for uppercase letters.
- **lower**: The character to mask for lowercase letters.
- **number**: The character to mask for numbers.

MASK_FIRST_N(str [, n])

Returns a masked version of str with the first n values masked. Upper case letters are converted to "X", lower case letters are converted to "x" and numbers are converted to "n". For example, mask_first_n("1234-5678-8765-4321", 4) results in nnnn-5678-8765-4321.

- **str**: The string to mask.
- **n**: The number of values to mask.

MASK_LAST_N(str [, n])

Returns a masked version of str with the last n values masked. Upper case letters are converted to "X", lower case letters are converted to "x" and numbers are converted to "n". For example, mask_last_n("1234-5678-8765-4321", 4) results in 1234-5678-8765-nnnn.

- **str**: The string to mask.
- **n**: The number of values to mask.

MASK_SHOW_FIRST_N(str [, n])

Returns a masked version of str, showing the first n characters unmasked. Upper case letters are converted to "X", lower case letters are converted to "x" and numbers are converted to "n". For example, mask_show_first_n("1234-5678-8765-4321", 4) results in 1234-nnnn-nnnn-nnnn.

- **str**: The string to mask.
- **n**: The number of values to mask.

MASK_SHOW_LAST_N(str [, n])

Returns a masked version of str, showing the last n characters unmasked. Upper case letters are converted to "X", lower case letters are converted to "x" and numbers are converted to "n". For example, mask_show_last_n("1234-5678-8765-4321", 4) results in nnnn-nnnn-nnnn-4321.

- **str**: The string to mask.
- **n**: The number of values to mask.

MASK_HASH(str)

Returns a hashed value based on str. The hash is consistent and can be used to join masked values together across tables. This function returns null for non-string types.

- **str**: The string to mask.

JAVA_METHOD(class, method [, arg1] [, ...])

Calls a method with reflection.

- **class**: The class to call.
- **method**: The method to call.
- **arg1**: The argument to pass in.

REFLECT(class, method [, arg1] [, ...])

Calls a method with reflection.

- **class**: The class to call.
- **method**: The method to call.
- **arg1**: The argument to pass in.

HASH(expr1 [, expr2] [, ...])

Returns a hash value of the arguments.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

CURRENT_USER()

Returns current user name from the configured authenticator manager. Could be the same as the user provided when connecting, but with some authentication managers (for example HadoopDefaultAuthenticator) it could be different.

LOGGED_IN_USER()

Returns current user name from the session state. This is the username provided when connecting to Hive.

CURRENT_DATABASE()

Returns current database name.

SHA1(expr)

Returns a sha1 hash value as a hex string of the expr.

- **expr**: A valid expression.

CRC32(expr)

Returns a cyclic redundancy check value of the expr as a bigint.

- **expr**: A valid expression.

SHA2(expr, bitlength)

Returns a checksum of SHA-2 family as a hex string of expr. SHA-224, SHA-256, SHA-384, and SHA-512 are supported. Bit length of 0 is equivalent to 256.

- **expr**: A valid expression.
- **bitlength**: The bit length.

AES_ENCRYPT(input, key)

Encrypt input using AES. Key lengths of 128, 192 or 256 bits can be used. 192 and 256 bits keys can be used if Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files are installed. If either argument is NULL or the key length is not one of the permitted values, the return value is NULL. Example: `base64(aes_encrypt('ABC', '1234567890123456')) = 'y6Ss+zCYObpCbgfWfyNWTw=='`.

- **input**: The input value to encrypt.
- **key**: The key to use when encrypting.

VERSION()

Returns the Hive version. The string contains 2 fields, the first being a build number and the second being a build hash. Example: "select version();" might return "2.1.0.2.5.0.0-1245r027527b9c5ce1a3d7d0b6d2e6de2378fb0c39232". Actual results will depend on your build.

COUNT(DISTINCT expr1 [, expr2] [, ...])

Returns the number of rows for which the supplied expression(s) are unique and non-null.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

SUM(expr)

Returns the sum calculated from values of a group.

- **expr**: A valid expression.

SUM(DISTINCT expr)

Returns the sum calculated from distinct values of a group.

- **expr**: A valid expression.

AVG(expr)

Returns the mean calculated from values of a group.

- **expr**: A valid expression.

AVG(DISTINCT expr)

Returns the mean calculated from distinct values of a group.

- **expr**: A valid expression.

MIN(expr)

Returns the minimum value of expr.

- **expr**: A valid expression.

MAX(expr)

Returns the maximum value of expr.

- **expr**: A valid expression.

VARIANCE(expr)

Returns the sample variance calculated from values of a group.

- **expr**: A valid expression.

STDDEV_POP(expr)

Returns the population standard deviation calculated from values of a group.

- **expr**: A valid expression.

STDDEV_SAMP(expr)

Returns the sample standard deviation calculated from values of a group.

- **expr**: A valid expression.

COVAR_POP(expr1, expr2)

Returns the population covariance of a set of number pairs.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

COVAR_SAMP(expr1, expr2)

Returns the sample covariance of a set of number pairs.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

CORR(expr1, expr2)

Returns Pearson coefficient of correlation between a set of number pairs.

- **expr1**: A valid expression.
- **expr2**: A valid expression.

PERCENTILE(col, percentage [, accuracy])

Returns the exact percentile value of numeric column col at the given percentage. The value of percentage must be between 0.0 and 1.0. The value of frequency should be positive integral

- **col**: A numeric expression.
- **percentage**: The percentage.
- **accuracy**: The accuracy to control approximation.

PERCENTILE_APPROX(col, percentage [, accuracy])

Returns the approximate percentile value of numeric column col at the given percentage. The value of percentage must be between 0.0 and 1.0. The accuracy parameter (default: 10000) is a positive numeric literal which controls approximation accuracy at the cost of memory. Higher value of accuracy yields better accuracy, 1.0/accuracy is the relative error of the approximation. When percentage is an array, each value of the percentage array must be between 0.0 and 1.0. In this case, returns the approximate percentile array of column col at the given percentage array.

- **col**: A numeric expression.
- **percentage**: The percentage.

- **accuracy**: The accuracy to control approximation.

COLLECT_SET(expr)

Collects and returns a set of unique elements.

- **expr**: A valid expression.

COLLECT_LIST(expr)

Collects and returns a set of unique elements.

- **expr**: A valid expression.

NTILE(n)

Divides the rows for each window partition into n buckets ranging from 1 to at most n.

- **n**: The number of buckets.

EXPLODE(expr)

Separates the elements of array expr into multiple rows, or the elements of map expr into multiple rows and columns.

- **expr**: A valid expression.

POSEXPLODE(expr)

Separates the elements of array expr into multiple rows with positions, or the elements of map expr into multiple rows and columns with positions.

- **expr**: A valid expression.

INLINE(expr)

Explodes an array of structs into a table.

- **expr**: A valid expression.

STACK(n, expr1 [, expr2] [, ...])

Separates expr1, ..., exprk into n rows.

- **n**: The number of rows.
- **expr1**: A valid expression.
- **expr2**: A valid expression.

PARSE_URL_TUPLE(urlStr, p1 [, p2] [, ...])

Takes URL string and a set of n URL parts, and returns a tuple of n values. This is similar to the `parse_url()` UDF but can extract multiple parts at once out of a URL. Valid part names are: HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, USERINFO, QUERY:[KEY].

- **urlStr**: A valid URL string.
- **p1**: A valid part name.
- **p2**: A valid part name.

SELECT INTO Statements

You can use the SELECT INTO statement to export formatted data to a file.

Data Export with an SQL Query

The following query exports data into a file formatted in comma-separated values (CSV):

```
boolean ret = stat.execute("SELECT City, CompanyName INTO
'csv://c:/Customers.txt' FROM 'Customers' WHERE Country = 'US'");
System.out.println(stat.getUpdateCount()+" rows affected");
```

You can specify other file formats in the URI. The following example exports tab-separated values:

```
Statement stat = conn.createStatement();
boolean ret = stat.execute("SELECT * INTO 'Customers' IN
```

```
'csv://filename=c:/Customers.csv;delimiter=tab' FROM "Customers" WHERE
Country = 'US');
System.out.println(stat.getUpdateCount()+" rows affected");
```

INSERT Statements

To create new records, use INSERT statements.

INSERT Syntax

The INSERT statement specifies the columns to be inserted and the new column values. You can specify the column values in a comma-separated list in the VALUES clause, as shown in the following example:

```
INSERT INTO <table_name>
( <column_reference> [ , ... ] )
VALUES
( { <expression> | NULL } [ , ... ] )

<expression> ::=
    | @ <parameter>
    | ?
    | <literal>
```

You can use the executeUpdate method of the Statement and PreparedStatement classes to execute data manipulation commands and retrieve the rows affected.

```
String cmd = "INSERT INTO Customers (CompanyName) VALUES (?)";
PreparedStatement pstmt = connection.prepareStatement(cmd);
pstmt.setString(1, "RSSBus Inc.");
int count = pstmt.executeUpdate();
System.out.println(count+" rows were affected");
connection.close();
```

EXECUTE Statements

To execute stored procedures, you can use EXECUTE or EXEC statements.

EXEC and EXECUTE assign stored procedure inputs, referenced by name, to values or parameter names.

Stored Procedure Syntax

To execute a stored procedure as an SQL statement, use the following syntax:

```
{ EXECUTE | EXEC } <stored_proc_name>
{
  [ @ ] <input_name> = <expression>
} [ , ... ]
<expression> ::=
  | @ <parameter>
  | ?
  | <literal>
```

Example Statements

Reference stored procedure inputs by name:

```
EXECUTE my_proc @second = 2, @first = 1, @third = 3;
```

Execute a parameterized stored procedure statement:

```
EXECUTE my_proc second = @p1, first = @p2, third = @p3;
```

PIVOT and UNPIVOT

PIVOT and **UNPIVOT** can be used to change a table-valued expression into another table.

PIVOT

PIVOT rotates a table-value expression by turning unique values from one column into multiple columns in the output. PIVOT can run aggregations where required on any column value.

PIVOT Syntax

```
"SELECT 'AverageCost' AS Cost_Sorted_By_Production_Days, [0], [1], [2],
[3], [4]"
```

```
FROM
(
  SELECT DaysToManufacture, StandardCost
  FROM Production.Product
) AS SourceTable
PIVOT
(
  AVG(StandardCost)
  FOR DaysToManufacture IN ([0], [1], [2], [3], [4])
) AS PivotTable;"
```

UNPIVOT

UNPIVOT carries out nearly the opposite to PIVOT by rotating columns of a table-valued expressions into column values.

UNPIVOT Syntax

```
"SELECT VendorID, Employee, Orders
FROM
(SELECT VendorID, Emp1, Emp2, Emp3, Emp4, Emp5
FROM pvt) p
UNPIVOT
(Orders FOR Employee IN
(Emp1, Emp2, Emp3, Emp4, Emp5)
)AS unpvt;"
```

For further information on PIVOT and UNPIVOT, see [FROM clause plus JOIN, APPLY, PIVOT \(Transact-SQL\)](#)

Data Model

The adapter leverages Spark Thrift to enable bidirectional SQL access to SparkSQL data. Spark version 1.6 and above are supported.

Discovering Schemas

The SparkSQL Adapter dynamically obtains the SparkSQL schemas; reconnect to pick up any metadata changes, such as added or removed columns or changes in data type.

Connection String Options

The connection string properties are the various options that can be used to establish a connection. This section provides a complete list of the options you can configure in the connection string for this provider. Click the links for further details.

For more information on establishing a connection, see [Basic Tab](#).

Authentication

Property	Description
AuthScheme	The authentication scheme used. Accepted entries are Plain, LDAP, NOSASL, and Kerberos.
Server	The host name or IP address of the server hosting the SparkSQL database.
Port	The port for the SparkSQL database.
User	The username used to authenticate with SparkSQL.
Password	The password used to authenticate with SparkSQL.
Database	The name of the SparkSQL database.
ProtocolVersion	The Protocol Version used to authenticate with SparkSQL.
ImpersonationProxyUser	The proxy user of the Hive user impersonation.
SaslQop	Quality of protection for the SASL framework. The level of quality is negotiated between the client and server during authentication. Used by Kerberos authentication with TCP transport.
TransportMode	The transport mode to use to communicate with the Hive server. Accepted entries are BINARY and HTTP.

Kerberos

Property	Description
KerberosKDC	The Kerberos Key Distribution Center (KDC) service used to authenticate the user.
KerberosRealm	The Kerberos Realm used to authenticate the user.
KerberosSPN	The service principal name (SPN) for the Kerberos Domain Controller.
KerberosKeytabFile	The Keytab file containing your pairs of Kerberos principals and encrypted keys.
KerberosServiceRealm	The Kerberos realm of the service.
KerberosServiceKDC	The Kerberos KDC of the service.
KerberosTicketCache	The full file path to an MIT Kerberos credential cache file.

SSL

Property	Description
SSLClientCert	The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).
SSLClientCertType	The type of key store containing the TLS/SSL client certificate.
SSLClientCertPassword	The password for the TLS/SSL client certificate.
SSLClientCertSubject	The subject of the TLS/SSL client certificate.
SSLServerCert	The certificate to be accepted from the server when connecting using TLS/SSL.

Firewall

Property	Description
FirewallType	The protocol used by a proxy-based firewall.
FirewallServer	The name or IP address of a proxy-based firewall.
FirewallPort	The TCP port for a proxy-based firewall.
FirewallUser	The user name to use to authenticate with a proxy-based firewall.
FirewallPassword	A password used to authenticate to a proxy-based firewall.

Proxy

Property	Description
ProxyAutoDetect	This indicates whether to use the system proxy settings or not. This takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.
ProxyServer	The hostname or IP address of a proxy to route HTTP traffic through.
ProxyPort	The TCP port the ProxyServer proxy is running on.
ProxyAuthScheme	The authentication type to use to authenticate to the ProxyServer proxy.
ProxyUser	A user name to be used to authenticate to the ProxyServer proxy.
ProxyPassword	A password to be used to authenticate to the ProxyServer proxy.
ProxySSLType	The SSL type to use when connecting to the ProxyServer proxy.
ProxyExceptions	A semicolon separated list of destination hostnames or IPs that are exempt from connecting through the ProxyServer .

Logging

Property	Description
LogModules	Core modules to be included in the log file.

Schema

Property	Description
Location	A path to the directory that contains the schema files defining tables, views, and stored procedures.

Miscellaneous

Property	Description
AsyncQueryTimeout	The timeout for asynchronous requests issued by the provider to download large result sets.
DescribeCommand	The describe command to determine which describe command will use to communicate with the Hive server. Accepted entries are DESCRIBE and DESC.
DetectView	Specifies whether to use DESCRIBE FORMATTED ... to detect the specified table is view or not.
HTTPPath	The path component of the URL endpoint when using HTTP TransportMode.
MaxRows	Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.
Other	These hidden properties are used only in specific use cases.

Readonly	You can use this property to enforce read-only access to Spark SQL from the provider.
ServerConfigurations	A name-value list of server configuration variables to override the server defaults.
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
UseDatabricksUploadApi	This option specifies whether the Databricks Upload API will be use when executing batch insert.
UseDescTableQuery	This option specifies whether the columns will be retrieved using a DESC TABLE query or the GetColumns Thrift API. The GetColumns Thrift API works for the SparkSQL 3.0.0 or later.
UseInsertSelectSyntax	Specifies whether to use an INSERT INTO SELECT statement.
UserDefinedViews	A filepath pointing to the JSON configuration file containing your custom views.
UseSSL	Specifies whether to use SSL Encryption when connecting to Hive.

Authentication

This section provides a complete list of the Authentication properties you can configure in the connection string for this provider.

Property	Description
AuthScheme	The authentication scheme used. Accepted entries are Plain, LDAP, NOSASL, and Kerberos.
Server	The host name or IP address of the server hosting the SparkSQL database.

Port	The port for the SparkSQL database.
User	The username used to authenticate with SparkSQL.
Password	The password used to authenticate with SparkSQL.
Database	The name of the SparkSQL database.
ProtocolVersion	The Protocol Version used to authenticate with SparkSQL.
ImpersonationProxyUser	The proxy user of the Hive user impersonation.
SaslQop	Quality of protection for the SASL framework. The level of quality is negotiated between the client and server during authentication. Used by Kerberos authentication with TCP transport.
TransportMode	The transport mode to use to communicate with the Hive server. Accepted entries are BINARY and HTTP.

AuthScheme

The authentication scheme used. Accepted entries are Plain, LDAP, NOSASL, and Kerberos.

Possible Values

NOSASL, Plain, LDAP, Kerberos

Data Type

string

Default Value

"Plain"

Remarks

The AuthScheme used to authenticate with SparkSQL.

Server

The host name or IP address of the server hosting the SparkSQL database.

Data Type

string

Default Value

""

Remarks

The host name or IP address of the server hosting the SparkSQL database.

Port

The port for the SparkSQL database.

Data Type

string

Default Value

"10000"

Remarks

The port for the SparkSQL database.

User

The username used to authenticate with SparkSQL.

Data Type

string

Default Value

""

Remarks

The username used to authenticate with SparkSQL.

Password

The password used to authenticate with SparkSQL.

Data Type

string

Default Value

""

Remarks

The password used to authenticate with SparkSQL.

Database

The name of the SparkSQL database.

Data Type

string

Default Value

""

Remarks

The name of the SparkSQL database.

ProtocolVersion

The Protocol Version used to authenticate with SparkSQL.

Possible Values

1, 2, 3, 4, 5, 6, 7, 8

Data Type

string

Default Value

"8"

Remarks

The Protocol Version used to authenticate with SparkSQL.

ImpersonationProxyUser

The proxy user of the Hive user impersonation.

Data Type

string

Default Value

""

Remarks

The proxy user of the Hive user impersonation.

SaslQop

Quality of protection for the SASL framework. The level of quality is negotiated between the client and server during authentication. Used by Kerberos authentication with TCP transport.

Possible Values

auth, auth-int, auth-conf

Data Type

string

Default Value

"auth"

Remarks

This property should be set to the 'hive.server2.thrift.sasl.qop' value specified in your Hive configuration file (hive-site.xml).

auth	Authentication only
------	---------------------

auth-int	Authentication plus integrity protection
auth-conf	Authentication plus integrity and confidentiality protection

TransportMode

The transport mode to use to communicate with the Hive server. Accepted entries are BINARY and HTTP.

Possible Values

BINARY, HTTP

Data Type

string

Default Value

"BINARY"

Remarks

The transport mode used to communicate with Hive server.

This property should be set to the 'hive.server2.transport.mode' value specified in your Hive configuration file (hive-site.xml).

Kerberos

This section provides a complete list of the Kerberos properties you can configure in the connection string for this provider.

Property	Description
----------	-------------

KerberosKDC	The Kerberos Key Distribution Center (KDC) service used to authenticate the user.
KerberosRealm	The Kerberos Realm used to authenticate the user.
KerberosSPN	The service principal name (SPN) for the Kerberos Domain Controller.
KerberosKeytabFile	The Keytab file containing your pairs of Kerberos principals and encrypted keys.
KerberosServiceRealm	The Kerberos realm of the service.
KerberosServiceKDC	The Kerberos KDC of the service.
KerberosTicketCache	The full file path to an MIT Kerberos credential cache file.

KerberosKDC

The Kerberos Key Distribution Center (KDC) service used to authenticate the user.

Data Type

string

Default Value

""

Remarks

The Kerberos properties are used when using SPNEGO or Windows Authentication. The adapter will request session tickets and temporary session keys from the Kerberos KDC service. The Kerberos KDC service is conventionally colocated with the domain controller.

If Kerberos KDC is not specified, the adapter will attempt to detect these properties automatically from the following locations:

- **KRB5 Config File (krb5.ini/krb5.conf):** If the KRB5_CONFIG environment variable is set and the file exists, the adapter will obtain the KDC from the specified file. Otherwise, it will attempt to read from the default MIT location based on the OS: `C:\ProgramData\MIT\Kerberos5\krb5.ini` (Windows) or `/etc/krb5.conf` (Linux).
- **Java System Properties:** Using the system properties `java.security.krb5.realm` and `java.security.krb5.kdc`.
- **Domain Name and Host:** If the Kerberos Realm and Kerberos KDC could not be inferred from another location, the adapter will infer them from the configured domain name and host.

Note: Windows authentication is supported in JRE 1.6 and above only.

KerberosRealm

The Kerberos Realm used to authenticate the user.

Data Type

string

Default Value

""

Remarks

The Kerberos properties are used when using SPNEGO or Windows Authentication. The Kerberos Realm is used to authenticate the user with the Kerberos Key Distribution Service (KDC). The Kerberos Realm can be configured by an administrator to be any string, but conventionally it is based on the domain name.

If Kerberos Realm is not specified, the adapter will attempt to detect these properties automatically from the following locations:

- **KRB5 Config File (krb5.ini/krb5.conf):** If the KRB5_CONFIG environment variable is set and the file exists, the adapter will obtain the default realm from the specified file. Otherwise, it will attempt to read from the default MIT location based on the OS: `C:\ProgramData\MIT\Kerberos5\krb5.ini` (Windows) or `/etc/krb5.conf` (Linux)
- **Java System Properties:** Using the system properties `java.security.krb5.realm` and

java.security.krb5.kdc.

- **Domain Name and Host:** If the Kerberos Realm and Kerberos KDC could not be inferred from another location, the adapter will infer them from the user-configured domain name and host. This might work in some Windows environments.

Note: Kerberos-based authentication is supported in JRE 1.6 and above only.

KerberosSPN

The service principal name (SPN) for the Kerberos Domain Controller.

Data Type

string

Default Value

""

Remarks

If the SPN on the Kerberos Domain Controller is not the same as the URL that you are authenticating to, use this property to set the SPN.

KerberosKeytabFile

The Keytab file containing your pairs of Kerberos principals and encrypted keys.

Data Type

string

Default Value

""

Remarks

The Keytab file containing your pairs of Kerberos principals and encrypted keys.

KerberosServiceRealm

The Kerberos realm of the service.

Data Type

string

Default Value

""

Remarks

The KerberosServiceRealm is the specify the service Kerberos realm when using cross-realm Kerberos authentication.

In most cases, a single realm and KDC machine are used to perform the Kerberos authentication and this property is not required.

This property is available for complex setups where a different realm and KDC machine are used to obtain an authentication ticket (AS request) and a service ticket (TGS request).

KerberosServiceKDC

The Kerberos KDC of the service.

Data Type

string

Default Value

""

Remarks

The `KerberosServiceKDC` is used to specify the service Kerberos KDC when using cross-realm Kerberos authentication.

In most cases, a single realm and KDC machine are used to perform the Kerberos authentication and this property is not required.

This property is available for complex setups where a different realm and KDC machine are used to obtain an authentication ticket (AS request) and a service ticket (TGS request).

KerberosTicketCache

The full file path to an MIT Kerberos credential cache file.

Data Type

string

Default Value

""

Remarks

This property can be set if you wish to use a credential cache file that was created using the MIT Kerberos Ticket Manager or kinit command.

SSL

This section provides a complete list of the SSL properties you can configure in the connection string for this provider.

Property	Description
SSLClientCert	The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).

SSLClientCertType	The type of key store containing the TLS/SSL client certificate.
SSLClientCertPassword	The password for the TLS/SSL client certificate.
SSLClientCertSubject	The subject of the TLS/SSL client certificate.
SSLServerCert	The certificate to be accepted from the server when connecting using TLS/SSL.

SSLClientCert

The TLS/SSL client certificate store for SSL Client Authentication (2-way SSL).

Data Type

string

Default Value

""

Remarks

The name of the certificate store for the client certificate.

The [SSLClientCertType](#) field specifies the type of the certificate store specified by [SSLClientCert](#). If the store is password protected, specify the password in [SSLClientCertPassword](#).

[SSLClientCert](#) is used in conjunction with the [SSLClientCertSubject](#) field in order to specify client certificates. If [SSLClientCert](#) has a value, and [SSLClientCertSubject](#) is set, a search for a certificate is initiated. See [SSLClientCertSubject](#) for more information.

Designations of certificate stores are platform-dependent.

The following are designations of the most common User and Machine certificate stores in Windows:

MY	A certificate store holding personal certificates with their associated private keys.
CA	Certifying authority certificates.
ROOT	Root certificates.
SPC	Software publisher certificates.

In Java, the certificate store normally is a file containing certificates and optional private keys.

When the certificate store type is PFXFile, this property must be set to the name of the file. When the type is PFXBlob, the property must be set to the binary contents of a PFX file (for example, PKCS12 certificate store).

SSLClientCertType

The type of key store containing the TLS/SSL client certificate.

Possible Values

USER, MACHINE, PFXFILE, PFXBLOB, JKSFILE, JKSLOB, PEMKEY_FILE, PEMKEY_BLOB, PUBLIC_KEY_FILE, PUBLIC_KEY_BLOB, SSHPUBLIC_KEY_FILE, SSHPUBLIC_KEY_BLOB, P7BFILE, PPKFILE, XMLFILE, XMLBLOB

Data Type

string

Default Value

"USER"

Remarks

This property can take one of the following values:

USER - default	For Windows, this specifies that the certificate store is a certificate store owned by the current user. Note that this store type is not available in Java.
MACHINE	For Windows, this specifies that the certificate store is a machine store. Note that this store type is not available in Java.
PFXFILE	The certificate store is the name of a PFX (PKCS12) file containing certificates.
PFXBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in PFX (PKCS12) format.
JKSFILE	The certificate store is the name of a Java key store (JKS) file containing certificates. Note that this store type is only available in Java.
JKSBLOB	The certificate store is a string (base-64-encoded) representing a certificate store in JKS format. Note that this store type is only available in Java.
PEMKEY_FILE	The certificate store is the name of a PEM-encoded file that contains a private key and an optional certificate.
PEMKEY_BLOB	The certificate store is a string (base64-encoded) that contains a private key and an optional certificate.
PUBLIC_KEY_FILE	The certificate store is the name of a file that contains a PEM- or DER-encoded public key certificate.
PUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains a PEM- or DER-encoded public key certificate.
SSHPUBLIC_KEY_FILE	The certificate store is the name of a file that contains an SSH-style public key.
SSHPUBLIC_KEY_BLOB	The certificate store is a string (base-64-encoded) that contains an SSH-style public key.
P7BFILE	The certificate store is the name of a PKCS7 file containing

	certificates.
PPKFILE	The certificate store is the name of a file that contains a PuTTY Private Key (PPK).
XMLFILE	The certificate store is the name of a file that contains a certificate in XML format.
XMLBLOB	The certificate store is a string that contains a certificate in XML format.

SSLClientCertPassword

The password for the TLS/SSL client certificate.

Data Type

string

Default Value

""

Remarks

If the certificate store is of a type that requires a password, this property is used to specify that password to open the certificate store.

SSLClientCertSubject

The subject of the TLS/SSL client certificate.

Data Type

string

Default Value

"*"

Remarks

When loading a certificate the subject is used to locate the certificate in the store.

If an exact match is not found, the store is searched for subjects containing the value of the property. If a match is still not found, the property is set to an empty string, and no certificate is selected.

The special value "*" picks the first certificate in the certificate store.

The certificate subject is a comma separated list of distinguished name fields and values. For example, "CN=www.server.com, OU=test, C=US, E=support@company.com". The common fields and their meanings are shown below.

Field	Meaning
CN	Common Name. This is commonly a host name like www.server.com.
O	Organization
OU	Organizational Unit
L	Locality
S	State
C	Country
E	Email Address

If a field value contains a comma, it must be quoted.

SSLServerCert

The certificate to be accepted from the server when connecting using TLS/SSL.

Data Type

string

Default Value

""

Remarks

If using a TLS/SSL connection, this property can be used to specify the TLS/SSL certificate to be accepted from the server. Any other certificate that is not trusted by the machine is rejected.

This property can take the following forms:

Description	Example
A full PEM Certificate (example shortened for brevity)	-----BEGIN CERTIFICATE----- MIICHTCCAe4CAQAwDQYJKoZIhvd.....Qw == -----END CERTIFICATE-----
A path to a local file containing the certificate	C:\cert.cer
The public key (example shortened for brevity)	-----BEGIN RSA PUBLIC KEY----- MIGfMA0GCSq.....AQAB -----END RSA PUBLIC KEY-----
The MD5 Thumbprint (hex values can also be either space or colon separated)	34a929226ae0819f2ec14b4a3d904f801cbb150d
The SHA1 Thumbprint (hex values can also be either space or colon separated)	34a929226ae0819f2ec14b4a3d904f801cbb150d

If not specified, any certificate trusted by the machine is accepted.

Certificates are validated as trusted by the machine based on the System's trust store. The trust store used is the 'javax.net.ssl.trustStore' value specified for the system. If no value is

specified for this property, Java's default trust store is used (for example, JAVA_HOME\lib\security\cacerts).

Use '*' to signify to accept all certificates. Note that this is not recommended due to security concerns.

Firewall

This section provides a complete list of the Firewall properties you can configure in the connection string for this provider.

Property	Description
FirewallType	The protocol used by a proxy-based firewall.
FirewallServer	The name or IP address of a proxy-based firewall.
FirewallPort	The TCP port for a proxy-based firewall.
FirewallUser	The user name to use to authenticate with a proxy-based firewall.
FirewallPassword	A password used to authenticate to a proxy-based firewall.

FirewallType

The protocol used by a proxy-based firewall.

Possible Values

NONE, TUNNEL, SOCKS4, SOCKS5

Data Type

string

Default Value

"NONE"

Remarks

This property specifies the protocol that the adapter will use to tunnel traffic through the [FirewallServer](#) proxy. Note that by default, the adapter connects to the system proxy; to disable this behavior and connect to one of the following proxy types, set [ProxyAutoDetect](#) to false.

Type	Default Port	Description
TUNNEL	80	When this is set, the adapter opens a connection to SparkSQL and traffic flows back and forth through the proxy.
SOCKS4	1080	When this is set, the adapter sends data through the SOCKS 4 proxy specified by FirewallServer and FirewallPort and passes the FirewallUser value to the proxy, which determines if the connection request should be granted.
SOCKS5	1080	When this is set, the adapter sends data through the SOCKS 5 proxy specified by FirewallServer and FirewallPort . If your proxy requires authentication, set FirewallUser and FirewallPassword to credentials the proxy recognizes.

To connect to HTTP proxies, use [ProxyServer](#) and [ProxyPort](#). To authenticate to HTTP proxies, use [ProxyAuthScheme](#), [ProxyUser](#), and [ProxyPassword](#).

FirewallServer

The name or IP address of a proxy-based firewall.

Data Type

string

Default Value

""

Remarks

This property specifies the IP address, DNS name, or host name of a proxy allowing traversal of a firewall. The protocol is specified by [FirewallType](#): Use [FirewallServer](#) with this property to connect through SOCKS or do tunneling. Use [ProxyServer](#) to connect to an HTTP proxy.

Note that the adapter uses the system proxy by default. To use a different proxy, set [ProxyAutoDetect](#) to false.

FirewallPort

The TCP port for a proxy-based firewall.

Data Type

int

Default Value

0

Remarks

This specifies the TCP port for a proxy allowing traversal of a firewall. Use [FirewallServer](#) to specify the name or IP address. Specify the protocol with [FirewallType](#).

FirewallUser

The user name to use to authenticate with a proxy-based firewall.

Data Type

string

Default Value

""

Remarks

The [FirewallUser](#) and [FirewallPassword](#) properties are used to authenticate against the proxy specified in [FirewallServer](#) and [FirewallPort](#), following the authentication method specified in [FirewallType](#).

FirewallPassword

A password used to authenticate to a proxy-based firewall.

Data Type

string

Default Value

""

Remarks

This property is passed to the proxy specified by [FirewallServer](#) and [FirewallPort](#), following the authentication method specified by [FirewallType](#).

Proxy

This section provides a complete list of the Proxy properties you can configure in the connection string for this provider.

Property	Description
ProxyAutoDetect	This indicates whether to use the system proxy settings or not. This

	takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.
ProxyServer	The hostname or IP address of a proxy to route HTTP traffic through.
ProxyPort	The TCP port the ProxyServer proxy is running on.
ProxyAuthScheme	The authentication type to use to authenticate to the ProxyServer proxy.
ProxyUser	A user name to be used to authenticate to the ProxyServer proxy.
ProxyPassword	A password to be used to authenticate to the ProxyServer proxy.
ProxySSLType	The SSL type to use when connecting to the ProxyServer proxy.
ProxyExceptions	A semicolon separated list of destination hostnames or IPs that are exempt from connecting through the ProxyServer .

ProxyAutoDetect

This indicates whether to use the system proxy settings or not. This takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.

Data Type

bool

Default Value

true

Remarks

This takes precedence over other proxy settings, so you'll need to set ProxyAutoDetect to FALSE in order use custom proxy settings.

NOTE: When this property is set to True, the proxy used is determined as follows:

- A search from the JVM properties (**http.proxy**, **https.proxy**, **socksProxy**, etc.) is performed.
- In the case that the JVM properties don't exist, a search from **java.home/lib/net.properties** is performed.
- In the case that `java.net.useSystemProxies` is set to True, a search from **the SystemProxy** is performed.
- In Windows only, an attempt is made to retrieve these properties from the **Internet Options** in the **registry**.

To connect to an HTTP proxy, see [ProxyServer](#). For other proxies, such as SOCKS or tunneling, see [FirewallType](#).

ProxyServer

The hostname or IP address of a proxy to route HTTP traffic through.

Data Type

string

Default Value

""

Remarks

The hostname or IP address of a proxy to route HTTP traffic through. The adapter can use the HTTP, Windows (NTLM), or Kerberos authentication types to authenticate to an HTTP proxy.

If you need to connect through a SOCKS proxy or tunnel the connection, see [FirewallType](#).

By default, the adapter uses the system proxy. If you need to use another proxy, set [ProxyAutoDetect](#) to false.

ProxyPort

The TCP port the ProxyServer proxy is running on.

Data Type

int

Default Value

80

Remarks

The port the HTTP proxy is running on that you want to redirect HTTP traffic through. Specify the HTTP proxy in [ProxyServer](#). For other proxy types, see [FirewallType](#).

ProxyAuthScheme

The authentication type to use to authenticate to the ProxyServer proxy.

Possible Values

BASIC, DIGEST, NONE, NEGOTIATE, NTLM, PROPRIETARY

Data Type

string

Default Value

"BASIC"

Remarks

This value specifies the authentication type to use to authenticate to the HTTP proxy specified by [ProxyServer](#) and [ProxyPort](#).

Note that the adapter will use the system proxy settings by default, without further configuration needed; if you want to connect to another proxy, you will need to set

[ProxyAutoDetect](#) to false, in addition to [ProxyServer](#) and [ProxyPort](#). To authenticate, set [ProxyAuthScheme](#) and set [ProxyUser](#) and [ProxyPassword](#), if needed.

The authentication type can be one of the following:

- **BASIC:** The adapter performs HTTP BASIC authentication.
- **DIGEST:** The adapter performs HTTP DIGEST authentication.
- **NEGOTIATE:** The adapter retrieves an NTLM or Kerberos token based on the applicable protocol for authentication.
- **PROPRIETARY:** The adapter does not generate an NTLM or Kerberos token. You must supply this token in the Authorization header of the HTTP request.

If you need to use another authentication type, such as SOCKS 5 authentication, see [FirewallType](#).

ProxyUser

A user name to be used to authenticate to the ProxyServer proxy.

Data Type

string

Default Value

""

Remarks

The [ProxyUser](#) and [ProxyPassword](#) options are used to connect and authenticate against the HTTP proxy specified in [ProxyServer](#).

You can select one of the available authentication types in [ProxyAuthScheme](#). If you are using HTTP authentication, set this to the user name of a user recognized by the HTTP proxy. If you are using Windows or Kerberos authentication, set this property to a user name in one of the following formats:

```
user@domain
domain\user
```

ProxyPassword

A password to be used to authenticate to the ProxyServer proxy.

Data Type

string

Default Value

""

Remarks

This property is used to authenticate to an HTTP proxy server that supports NTLM (Windows), Kerberos, or HTTP authentication. To specify the HTTP proxy, you can set [ProxyServer](#) and [ProxyPort](#). To specify the authentication type, set [ProxyAuthScheme](#).

If you are using HTTP authentication, additionally set [ProxyUser](#) and [ProxyPassword](#) to HTTP proxy.

If you are using NTLM authentication, set [ProxyUser](#) and [ProxyPassword](#) to your Windows password. You may also need these to complete Kerberos authentication.

For SOCKS 5 authentication or tunneling, see [FirewallType](#).

By default, the adapter uses the system proxy. If you want to connect to another proxy, set [ProxyAutoDetect](#) to false.

ProxySSLType

The SSL type to use when connecting to the ProxyServer proxy.

Possible Values

AUTO, ALWAYS, NEVER, TUNNEL

Data Type

string

Default Value

"AUTO"

Remarks

This property determines when to use SSL for the connection to an HTTP proxy specified by [ProxyServer](#). This value can be AUTO, ALWAYS, NEVER, or TUNNEL. The applicable values are the following:

AUTO	Default setting. If the URL is an HTTPS URL, the adapter will use the TUNNEL option. If the URL is an HTTP URL, the component will use the NEVER option.
ALWAYS	The connection is always SSL enabled.
NEVER	The connection is not SSL enabled.
TUNNEL	The connection is through a tunneling proxy. The proxy server opens a connection to the remote host and traffic flows back and forth through the proxy.

ProxyExceptions

A semicolon separated list of destination hostnames or IPs that are exempt from connecting through the ProxyServer .

Data Type

string

Default Value

""

Remarks

The [ProxyServer](#) is used for all addresses, except for addresses defined in this property. Use semicolons to separate entries.

Note that the adapter uses the system proxy settings by default, without further configuration needed; if you want to explicitly configure proxy exceptions for this connection, you need to set [ProxyAutoDetect](#) = false, and configure [ProxyServer](#) and [ProxyPort](#). To authenticate, set [ProxyAuthScheme](#) and set [ProxyUser](#) and [ProxyPassword](#), if needed.

Logging

This section provides a complete list of the Logging properties you can configure in the connection string for this provider.

Property	Description
LogModules	Core modules to be included in the log file.

LogModules

Core modules to be included in the log file.

Data Type

string

Default Value

""

Remarks

Only the modules specified (separated by ';') will be included in the log file. By default all modules are included.

See the [Logging](#) page for an overview.

Schema

This section provides a complete list of the Schema properties you can configure in the connection string for this provider.

Property	Description
Location	A path to the directory that contains the schema files defining tables, views, and stored procedures.

Location

A path to the directory that contains the schema files defining tables, views, and stored procedures.

Data Type

string

Default Value

"%APPDATA%\CDData\SparkSQL Data Provider\Schema"

Remarks

The path to a directory which contains the schema files for the adapter (.rsd files for tables and views, .rsb files for stored procedures). The folder location can be a relative path from the location of the executable. The [Location](#) property is only needed if you want to customize definitions (for example, change a column name, ignore a column, and so on) or extend the data model with new tables, views, or stored procedures.

If left unspecified, the default location is "%APPDATA%\CDData\SparkSQL Data Provider\Schema" with **%APPDATA%** being set to the user's configuration directory:

Platform	%APPDATA%
Windows	The value of the APPDATA environment variable
Mac	~/Library/Application Support
Linux	~/.config

Miscellaneous

This section provides a complete list of the Miscellaneous properties you can configure in the connection string for this provider.

Property	Description
AsyncQueryTimeout	The timeout for asynchronous requests issued by the provider to download large result sets.
DescribeCommand	The describe command to determine which describe command will use to communicate with the Hive server. Accepted entries are DESCRIBE and DESC.
DetectView	Specifies whether to use DESCRIBE FORMATTED ... to detect the specified table is view or not.
HTTPPath	The path component of the URL endpoint when using HTTP TransportMode.
MaxRows	Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.
Other	These hidden properties are used only in specific use cases.
Readonly	You can use this property to enforce read-only access to Spark SQL from the provider.

ServerConfigurations	A name-value list of server configuration variables to override the server defaults.
Timeout	The value in seconds until the timeout error is thrown, canceling the operation.
UseDatabricksUploadApi	This option specifies whether the Databricks Upload API will be use when executing batch insert.
UseDescTableQuery	This option specifies whether the columns will be retrieved using a DESC TABLE query or the GetColumns Thrift API. The GetColumns Thrift API works for the SparkSQL 3.0.0 or later.
UseInsertSelectSyntax	Specifies whether to use an INSERT INTO SELECT statement.
UserDefinedViews	A filepath pointing to the JSON configuration file containing your custom views.
UseSSL	Specifies whether to use SSL Encryption when connecting to Hive.

AsyncQueryTimeout

The timeout for asynchronous requests issued by the provider to download large result sets.

Data Type

int

Default Value

300

Remarks

If the [AsyncQueryTimeout](#) property is set to 0, asynchronous operations will not time out; instead, they will run until they complete successfully or encounter an error condition. This

property is distinct from [Timeout](#) which applies to individual operations while [AsyncQueryTimeout](#) applies to execution time of the operation as a whole.

If [AsyncQueryTimeout](#) expires and the asynchronous request has not finished being processed, the adapter raises an error condition.

DescribeCommand

The describe command to determine which describe command will use to communicate with the Hive server. Accepted entries are DESCRIBE and DESC.

Possible Values

DESCRIBE, DESC

Data Type

string

Default Value

"DESCRIBE"

Remarks

DetectView

Specifies whether to use DESCRIBE FORMATTED ... to detect the specified table is view or not.

Data Type

bool

Default Value

true

Remarks

Specifies whether to use DESCRIBE FORMATTED ... to detect the specified table is view or not.

HTTPPath

The path component of the URL endpoint when using HTTP TransportMode.

Data Type

string

Default Value

"cliservice"

Remarks

This property is used to specify the path component of the URL endpoint when using HTTP [TransportMode](#).

This property should be set to the value specified in the 'hive.server2.thrift.http.path' property of you Hive configuration file (hive-site.xml).

MaxRows

Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

Data Type

int

Default Value

-1

Remarks

Limits the number of rows returned rows when no aggregation or group by is used in the query. This helps avoid performance issues at design time.

Other

These hidden properties are used only in specific use cases.

Data Type

string

Default Value

""

Remarks

The properties listed below are available for specific use cases. Normal driver use cases and functionality should not require these properties.

Specify multiple properties in a semicolon-separated list.

Integration and Formatting

DefaultColumnSize	Sets the default length of string fields when the data source does not provide column length in the metadata. The default value is 2000.
ConvertDateTimeToGMT	Determines whether to convert date-time values to GMT, instead of the local time of the machine.
RecordToFile=filename	Records the underlying socket data transfer to the specified file.

Readonly

You can use this property to enforce read-only access to Spark SQL from the provider.

Data Type

bool

Default Value

false

Remarks

If this property is set to true, the adapter will allow only SELECT queries. INSERT, UPDATE, DELETE, and stored procedure queries will cause an error to be thrown.

ServerConfigurations

A name-value list of server configuration variables to override the server defaults.

Data Type

string

Default Value

""

Remarks

This property takes a comma separated list of configuration variables specified as name-value pairs. Any values specified here will be sent to the Hive server to override the default values.

Example: hive.enforce.bucketing=true,hive.enforce.sorting=true

Timeout

The value in seconds until the timeout error is thrown, canceling the operation.

Data Type

int

Default Value

60

Remarks

If Timeout = 0, operations do not time out. The operations run until they complete successfully or until they encounter an error condition.

If Timeout expires and the operation is not yet complete, the adapter throws an exception.

UseDatabricksUploadApi

This option specifies whether the Databricks Upload API will be use when executing batch insert.

Data Type

bool

Default Value

false

Remarks

When set to true, the Databricks Upload API will be use when executing batch insert.

UseDescTableQuery

This option specifies whether the columns will be retrieved using a DESC TABLE query or the GetColumns Thrift API. The GetColumns Thrift API works for the SparkSQL 3.0.0 or later.

Data Type

bool

Default Value

true

Remarks

When set to true, a DESC TABLE query will be issued to retrieve the columns for the table.

UseInsertSelectSyntax

Specifies whether to use an INSERT INTO SELECT statement.

Data Type

bool

Default Value

false

Remarks

When set to true, an INSERT INTO SELECT statement will be used when executing insert statements. When set to false, an INSERT INTO VALUES statement will be used.

Unless explicitly specified, this option will be configured accordingly based on the Spark version.

UserDefinedViews

A filepath pointing to the JSON configuration file containing your custom views.

Data Type

string

Default Value

""

Remarks

User Defined Views are defined in a JSON-formatted configuration file called *UserDefinedViews.json*. The adapter automatically detects the views specified in this file.

You can also have multiple view definitions and control them using the UserDefinedViews connection property. When you use this property, only the specified views are seen by the adapter.

This User Defined View configuration file is formatted as follows:

- Each root element defines the name of a view.
- Each root element contains a child element, called **query**, which contains the custom SQL query for the view.

For example:

```
{
  "MyView": {
    "query": "SELECT * FROM Customers WHERE MyColumn = 'value'"
  },
  "MyView2": {
    "query": "SELECT * FROM MyTable WHERE Id IN (1,2,3)"
  }
}
```

Use the UserDefinedViews connection property to specify the location of your JSON configuration file. For example:

```
"UserDefinedViews",
"C:\\Users\\yourusername\\Desktop\\tmp\\UserDefinedViews.json"
```

UseSSL

Specifies whether to use SSL Encryption when connecting to Hive.

Data Type

bool

Default Value

false

Remarks

Set this property to the value specified in the 'hive.server2.use.SLL' property of your Hive configuration file (hive-site.xml).

SparkSQL Adapter Limitations

The results of the string functions SUBSTR and SUBSTRING might differ between pushed and not pushed if the second argument (length) is negative.

The syntax of these functions are:

```
SUBSTR (string, start_position, length_of_substring)
```

```
SUBSTRING (string, start_position, length_of_substring)
```

TIBCO Product Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join the TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [TIBCO Product Documentation](#) website, mainly in HTML and PDF formats.

The [TIBCO Product Documentation](#) website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for this product is available on the [TIBCO® Data Virtualization](#) page.

- **Users**
 - TDV Getting Started Guide
 - TDV User Guide
 - TDV Web UI User Guide
 - TDV Client Interfaces Guide
 - TDV Tutorial Guide
 - TDV Northbay Example
- **Administration**
 - TDV Installation and Upgrade Guide
 - TDV Administration Guide
 - TDV Active Cluster Guide
 - TDV Security Features Guide
- **Data Sources**

TDV Adapter Guides

TDV Data Source Toolkit Guide (Formerly Extensibility Guide)

- **References**

TDV Reference Guide

TDV Application Programming Interface Guide

- **Other**

TDV Business Directory Guide

TDV Discovery Guide

- *TIBCO TDV and Business Directory Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

How to Contact TIBCO Support

Get an overview of [TIBCO Support](#). You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the [TIBCO Support](#) website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to [TIBCO Support](#) website. If you do not have a user name, you can request one by clicking **Register** on the website.

Release Version Support

TDV 8.5 is designated as a Long Term Support (LTS) version. Some release versions of TIBCO® Data Virtualization products are selected to be long-term support (LTS) versions. Defect corrections will typically be delivered in a new release version and as hotfixes or service packs to one or more LTS versions. See also

https://docs.tibco.com/pub/tdv/general/LTS/tdv_LTS_releases.htm.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, visit [TIBCO Community](#).

Legal and Third-Party Notices

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, TIBCO logo, TIBCO O logo, ActiveSpaces, Enterprise Messaging Service, Spotfire, TERR, S-PLUS, and S+ are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the

readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 2002-2023 Cloud Software Group, Inc All Rights Reserved.